Package 'BayesForge'

November 26, 2026

Type Package

Title Bayesian Inference

Version 0.0.2

Description A high-performance probabilistic programming library that aims to unify the modeling experience by providing an intuitive model-building syntax together with the flexibility of low-level abstraction coding. It also includes pre-built functions for high-level abstraction and supports hardware-accelerated computation for improved scalability, including parallelization, vectorization, and execution on CPU, GPU, or TPU.

Encoding UTF-8

Imports reticulate (>= 1.45.0), abind, methods, parallel, future

SystemRequirements Python (>= 3.12), 'bayesforge' python library (installed by setup_env()).

Depends R (>= 3.6.0)

Suggests testthat (>= 3.0.0), posterior, brms, loo, knitr, bayesplot, rstantools, rstan

Config/testthat/edition 3

License GPL (>= 3)

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Sebastian Sosa [aut, cre]

Maintainer Sebastian Sosa <bf@s-sosa.com>

Repository CRAN

Date/Publication 2026-11-26 08:30:48 UTC

Contents

bf.dist.asymmetric_laplace 3
bf.dist.asymmetric_laplace_quantile 5
bf.dist.bernoulli 7
bf.dist.beta 9

bf.dist.beta_binomial . . . 11
bf.dist.beta_proportion . . . 14
bf.dist.binomial . . . 16
bf.dist.car . . . 18
bf.dist.categorical . . . 21
bf.dist.cauchy . . . 23
bf.dist.chi2 . . . 25
bf.dist.delta . . . 27
bf.dist.dirichlet . . . 29
bf.dist.dirichlet_multinomial . . . 31
bf.dist.discrete_uniform . . . 33
bf.dist.euler_maruyama . . . 35
bf.dist.exponential . . . 37
bf.dist.gamma . . . 39
bf.dist.gamma_poisson . . . 41
bf.dist.gaussian_copula . . . 43
bf.dist.gaussian_copula_beta . . . 45
bf.dist.gaussian_random_walk . . . 47
bf.dist.gaussian_state_space . . . 48
bf.dist.geometric . . . 50
bf.dist.gompertz . . . 52
bf.dist.gumbel . . . 54
bf.dist.half_cauchy . . . 56
bf.dist.half_normal . . . 58
bf.dist.inverse_gamma . . . 59
bf.dist.kumaraswamy . . . 61
bf.dist.laplace . . . 63
bf.dist.left_truncated_distribution . . . 65
bf.dist.levy . . . 67
bf.dist.lkj . . . 69
bf.dist.lkj_cholesky . . . 71
bf.dist.logistic . . . 73
bf.dist.log_normal . . . 75
bf.dist.log_uniform . . . 77
bf.dist.lower_truncated_power_law . . . 79
bf.dist.low_rank_multivariate_normal . . . 81
bf.dist.matrix_normal . . . 83
bf.dist.mixture . . . 86
bf.dist.mixture_general . . . 88
bf.dist.mixture_same_family . . . 91
bf.dist.multinomial . . . 93
bf.dist.multinomial_logits . . . 96
bf.dist.multinomial_probs . . . 98
bf.dist.multivariate_normal . . . 100
bf.dist.multivariate_student_t . . . 102
bf.dist.negative_binomial . . . 105
bf.dist.negative_binomial2 . . . 107
bf.dist.negative_binomial_logits . . . 109

bf.dist.negative_binomial_probs . . . 111
bf.dist.normal . . . 112
bf.dist.ordered_logistic . . . 114
bf.dist.pareto . . . 117
bf.dist.poisson . . . 119
bf.dist.projected_normal . . . 121
bf.dist.relaxed_bernoulli . . . 122
bf.dist.relaxed_bernoulli_logits . . . 125
bf.dist.right_truncated_distribution . . . 127
bf.dist.soft_laplace . . . 129
bf.dist.student_t . . . 131
bf.dist.truncated_cauchy . . . 133
bf.dist.truncated_distribution . . . 135
bf.dist.truncated_normal . . . 138
bf.dist.truncated_polya_gamma . . . 140
bf.dist.two_sided_truncated_distribution . . . 142
bf.dist.uniform . . . 144
bf.dist.weibull . . . 146
bf.dist.wishart . . . 148
bf.dist.wishart_cholesky . . . 150
bf.dist.zero_inflated_distribution . . . 152
bf.dist.zero_inflated_negative_binomial . . . 155
bf.dist.zero_inflated_poisson . . . 157
bf.dist.zero_sum_normal . . . 159
bf.doc . . . 161
bf_available . . . 162
bf_draws . . . 162
bf_model . . . 163
bf_plot_ns . . . 164
bf_sample . . . 165
bf_show . . . 166
importBF . . . 166
jax_modules . . . 167
loo.BayesForge.Main.main.bf . . . 167
setup_env . . . 168
to_brms . . . 169

Index **170**

`bf.dist.asymmetric_laplace`

Asymmetric Laplace distribution

Description

Samples from an Asymmetric Laplace distribution.

The Asymmetric Laplace distribution is a generalization of the Laplace distribution, where the two sides of the distribution are scaled differently. It is defined by a location parameter (loc), a scale parameter (scale), and an asymmetry parameter (asymmetry).

$$f(x,\kappa) = \begin{cases} \frac{1}{\kappa+\kappa^{-1}} \exp(-x\kappa), & x \geq 0, \\ \frac{1}{\kappa+\kappa^{-1}} \exp(x/\kappa), & x < 0 \end{cases}$$

for $-\infty < x < \infty$, $\kappa > 0$.

laplace_asymmetric takes 'kappa' as a shape parameter for κ. For $\kappa = 1$, it is identical to a Laplace distribution.

Usage

```
bf.dist.asymmetric_laplace(
  loc = 0,
  scale = 1,
  asymmetry = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	Location parameter of the distribution.
scale	Scale parameter of the distribution.
asymmetry	Asymmetry parameter of the distribution.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().

sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF AsymmetricLaplace distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the AsymmetricLaplace distribution (for direct sampling).
- When create_obj = TRUE: The raw BF AsymmetricLaplace distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#asymmetriclaplace

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.asymmetric_laplace(loc = 0.0, scale = 1.0, asymmetry = 1.0, sample = TRUE)
```

---

bf.dist.asymmetric_laplace_quantile

Asymmetric Laplace Quantile distribution

Description

Samples from an AsymmetricLaplaceQuantile distribution.

This distribution is an alternative parameterization of the AsymmetricLaplace distribution, commonly used in Bayesian quantile regression. It utilizes a quantile parameter to define the balance between the left- and right-hand sides of the distribution, representing the proportion of probability density that falls to the left-hand side.

Usage

```
bf.dist.asymmetric_laplace_quantile(
  loc = 0,
  scale = 1,
  quantile = 0.5,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	The location parameter of the distribution.
scale	The scale parameter of the distribution.
quantile	The quantile parameter, representing the proportion of probability density to the left of the median. Must be between 0 and 1.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When `sample = FALSE`: A BF AsymmetricLaplaceQuantile distribution object (for model building).
- When `sample = TRUE`: A JAX array of samples drawn from the AsymmetricLaplaceQuantile distribution (for direct sampling).
- When `create_obj = TRUE`: The raw BF AsymmetricLaplaceQuantile distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#asymmetriclaplacequantile

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.asymmetric_laplace_quantile(loc = 0.0, scale = 1.0, quantile = 0.5, sample = TRUE)
```

---

bf.dist.bernoulli *Bernoulli distribution*

Description

The Bernoulli distribution models a single trial with two possible outcomes: success or failureIt models a single trial with exactly two possible outcomes:

- **Success** (often coded as 1) with probability (p),
- **Failure** (often coded as 0) with probability (1 - p).

It is a building block for many more complex discrete distributions (e.g. binomial, geometric).

We write

$$X \sim \text{Bernoulli}(p)$$

with $0 \leq p \leq 1$.

Probability Mass Function (PMF):

$$P(X = x) = \begin{cases} p, & \text{if } x = 1, \\ 1 - p, & \text{if } x = 0, \\ 0, & \text{otherwise.} \end{cases}$$

Alternatively, using an "indicator" exponent form:

$$P(X = x) = p^x (1 - p)^{1-x}, \quad x \in 0, 1.$$

Usage

```
bf.dist.bernoulli(
  probs = py_none(),
  logits = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  infer = py_none(),
  enum = FALSE,
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

probs	Probability of success for each Bernoulli trial. Must be between 0 and 1.
logits	Log-odds of success for each Bernoulli trial. probs = sigmoid(logits).
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
infer	Optional named list passed to NumPyro's infer argument (e.g. list(enumerate = "parallel")).
enum	Logical. If TRUE, the discrete latent is summed out exactly (parallel enumeration) instead of sampled; same as infer = list(enumerate = "parallel"). Defaults to FALSE.
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).

`event`	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to `0`.
`create_obj`	Logical. If `TRUE`, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to `FALSE`.
`to_jax`	Logical. Return a JAX array (`TRUE`, default) or convert to R.

Value

- BF Bernoulli distribution object (for model building) when `sample = FALSE`.
- JAX array of samples drawn from the Bernoulli distribution (for direct sampling) when `sample = TRUE`.
- The raw BF distribution object (for advanced use cases) when `create_obj = TRUE`.

References

[1]: https://en.wikipedia.org/wiki/Bernoulli_distribution "Bernoulli distribution"

[2]: https://stats.libretexts.org/Courses/Saint_Mary\ "3.3: Bernoulli and Binomial Distributions - Statistics LibreTexts"

See Also

https://num.pyro.ai/en/stable/distributions.html#bernoulli

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.bernoulli(probs = 0.7, sample = TRUE)
```

`bf.dist.beta` *Beta distribution*

Description

Samples from a Beta distribution, defined on the interval [0, 1]. The Beta distribution is a versatile distribution often used to model probabilities or proportions. It is parameterized by two positive shape parameters, usually denoted α and $\beta > 0$, control the shape of the density (how much mass is pushed toward 0, 1, or intermediate).

$$X \sim Beta(\alpha, \beta), f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}, B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}, F(x) = I_x(\alpha+\beta)$$

where $B(\alpha, \beta)$ is the Beta function:

$$B(\alpha, \beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1}, dx = \frac{\Gamma(\alpha), \Gamma(\beta)}{\Gamma(\alpha+\beta)}.$$

where :numα and β are the concentration parameters, and $B(x, y)$ is the Beta function.

Usage

```
bf.dist.beta(
  concentration1,
  concentration0,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

concentration1	The first concentration parameter (shape parameter). Must be positive.
concentration0	The second concentration parameter (shape parameter). Must be positive.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.

`create_obj`	Logical. If `TRUE`, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to `FALSE`.
`to_jax`	Logical. Return a JAX array (`TRUE`, default) or convert to R.

Value

- When `sample = FALSE`: A BF Beta distribution object (for model building).
- When `sample = TRUE`: A JAX array of samples drawn from the Beta distribution (for direct sampling).
- When `create_obj = TRUE`: The raw BF Beta object (for advanced use cases).

References

[1]: https://en.wikipedia.org/wiki/Beta_distribution "Beta distribution - Wikipedia"

[2]: https://www.statlect.com/probability-distributions/beta-distribution "Beta distribution | Properties, proofs, exercises - StatLect"

[3]: https://reference.wolfram.com/language/ref/BetaDistribution.html "BetaDistribution - Wolfram Language Documentation"

See Also

https://num.pyro.ai/en/stable/distributions.html#beta

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.beta(concentration1 = 1.0, concentration0 = 1.0, sample = TRUE)
```

---

bf.dist.beta_binomial *BetaBinomial distribution*

Description

Samples from a BetaBinomial distribution, a compound distribution where the probability of success in a binomial experiment is drawn from a Beta distribution. This models situations where the underlying probability of success is not fixed but varies according to a prior belief represented by the Beta distribution. It is often used to model over-dispersion relative to the binomial distribution.

For $X \sim \text{BetaBinomial}(N, \alpha, \beta)$, for $k = 0, 1, \ldots, N$:

$$\Pr(X = k) = \binom{N}{k}; \frac{B(k + \alpha, ; N - k + \beta)}{B(\alpha, \beta)},$$

where ($B(u, v)$) is the Beta function ($B(u, v) = \frac{\Gamma(u),\Gamma(v)}{\Gamma(u+v)}$).

Usage

```
bf.dist.beta_binomial(
  concentration1,
  concentration0,
  total_count = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  infer = py_none(),
  enum = FALSE,
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

concentration1	The first concentration parameter (alpha) of the Beta distribution.
concentration0	The second concentration parameter (beta) of the Beta distribution.
total_count	The number of Bernoulli trials in the Binomial part of the distribution.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
infer	Optional named list passed to NumPyro's infer argument (e.g. list(enumerate = "parallel")).
enum	Logical. If TRUE, the discrete latent is summed out exactly (parallel enumeration) instead of sampled; same as infer = list(enumerate = "parallel"). Defaults to FALSE.
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.

shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF BetaBinomial distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the BetaBinomial distribution (for direct sampling).
- When create_obj = TRUE: The raw BF BetaBinomial distribution object (for advanced use cases).

References

[1]: https://en.wikipedia.org/wiki/Beta-binomial_distribution "Beta-binomial distribution"

[2]: https://www.statisticshowto.com/beta-binomial-distribution/ "Beta-Binomial Distribution: Definition - Statistics How To"

[3]: https://mc-stan.org/docs/2_19/functions-reference/beta-binomial-distribution.html "12.3 Beta-Binomial Distribution | Stan Functions Reference"

See Also

https://num.pyro.ai/en/stable/distributions.html#betabinomial

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.beta_binomial(
    concentration1 = 1.0,
    concentration0 = 1.0,
    total_count = 10L,
    sample = TRUE
)
```

`bf.dist.beta_proportion` *Beta Proportion distribution*

Description

The Beta Proportion distribution is a reparameterization of the conventional Beta distribution in terms of a the variate mean and a precision parameter. It's useful for modeling rates and proportions. It's essentially the same family as the standard Beta((\alpha,\beta)), but the mapping is:

$$\alpha = \mu, \kappa, \quad \beta = (1 - \mu), \kappa.$$

Thus the pdf is:

$$f(\theta \mid \mu, \kappa) = \frac{1}{B(\mu\kappa, , (1-\mu)\kappa)}; \theta^{\mu\kappa - 1}; (1-\theta)^{(1-\mu)\kappa - 1}, \theta \in (0, 1).$$

This parameterization is useful in hierarchical models, since one can put hyperpriors on (\mu) (the "center") and (\kappa) (the "spread").

Usage

```
bf.dist.beta_proportion(
  mean,
  concentration,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`mean`	The mean of the BetaProportion distribution,must be between 0 and 1.
`concentration`	The concentration parameter of the BetaProportion distribution.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.

obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF BetaProportion distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the BetaProportion distribution (for direct sampling).
- When create_obj = TRUE: The raw BF BetaProportion distribution object (for advanced use cases).

References

https://mc-stan.org/docs/2_19/functions-reference/beta-proportion-distribution.html "19.2 Beta Proportion Distribution | Stan Functions Reference"

See Also

https://num.pyro.ai/en/stable/distributions.html#beta_proportion

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
samples <- bf.dist.beta_proportion(
    mean = 0.5,
    concentration = 2.0,
    sample = TRUE,
    shape = c(1000L)
)
```

`bf.dist.binomial` *Binomial distribution*

Description

The Binomial distribution models the number of successes in a sequence of independent Bernoulli trials. It represents the probability of obtaining exactly *k* successes in *n* trials, where each trial has a probability *p* of success. It assumes:

1. Each trial has exactly two possible outcomes (success or failure).
2. The probability of success $p is constant across trials.
3. Trials are independent.

We denote:

$$X \sim \text{Binomial}(n, p).$$

Probability Mass Function (PMF):
For $k = 0, 1, 2, \ldots, n$:

$$\Pr(X = k) = \binom{n}{k}; p^k; (1-p)^{,n-k},$$

where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.
This formula counts the number of ways to choose which k out of n trials are the successes (the binomial coefficient) and then multiplies by the probability of each such configuration $p^k(1-p)^{n-k}$.

`bf.dist.bfnomial()` is a deprecated alias of `bf.dist.binomial()`.

Usage

```
bf.dist.binomial(
  total_count = 1L,
  probs = py_none(),
  logits = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  infer = py_none(),
  enum = FALSE,
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
```

```
  to_jax = TRUE
)

bf.dist.bfnomial(...)
```

Arguments

total_count	The number of trials n.
probs	The probability of success p for each trial. Must be between 0 and 1.
logits	The log-odds of success for each trial. probs = jax.nn.sigmoid(logits).
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
infer	Optional named list passed to NumPyro's infer argument (e.g. list(enumerate = "parallel")).
enum	Logical. If TRUE, the discrete latent is summed out exactly (parallel enumeration) instead of sampled; same as infer = list(enumerate = "parallel"). Defaults to FALSE.
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.
...	Arguments passed to bf.dist.binomial().

Value

- Binomial distribution object (for model building) when sample = FALSE.
- JAX array of samples drawn from the Binomial distribution (for direct sampling) when sample = TRUE.
- The raw BF distribution object (for advanced use cases) when create_obj = TRUE.

References

[1]: https://en.wikipedia.org/wiki/Binomial_distribution "Binomial distribution"

[2]: https://www.itl.nist.gov/div898/handbook/eda/section3/eda366i.htm "1.3.6.6.18. Binomial Distribution - Information Technology Laboratory"

See Also

https://num.pyro.ai/en/stable/distributions.html#binomial

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.binomial(total_count = 10L, probs = 0.5, sample = TRUE)
```

`bf.dist.car` *Conditional Autoregressive (CAR) distribution*

Description

The CAR distribution models a vector of variables where each variable is a linear combination of its neighbors in a graph. The CAR model captures spatial dependence in areal data by modeling each observation as conditionally dependent on its neighbors.It specifies a joint distribution of a vector of random variables $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ based on their conditional distributions, where each y_i is conditionally independent of all other variables given its neighbors.

- **Application**: Widely used in disease mapping, environmental modeling, and spatial econometrics to account for spatial autocorrelation.
- **Conditional Distribution**: Each y_i given its neighbors $\mathbf{y}_{-i}$ follows a normal distribution:

$$y_i|\mathbf{y} * -i \sim \mathcal{N}\left(\frac{\sum * j \in \mathcal{N}(i) w_{ij} y_j}{\sum_{j \in \mathcal{N}(i)} w_{ij}}, \sigma^2\right)$$

where:

- $\mathcal{N}(i)$ denotes the set of neighbors of unit i,
- w_{ij} are the weights representing the strength of the spatial relationship between units i and j,
- σ^2 is the variance term. where μ_i is a function of the values of the neighbors of site i and Σ_i is the variance of site i.

Usage

```
bf.dist.car(
  loc,
  correlation,
  conditional_precision,
  adj_matrix,
  is_sparse = FALSE,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	Mean of the distribution.
correlation	Correlation between variables.
conditional_precision	Precision of the distribution.
adj_matrix	Adjacency matrix defining the graph. is_sparse (bool): Whether the adjacency matrix is sparse. Defaults to FALSE. - *validate_args/ (bool): Whether to validate arguments. Defaults to py_none().
is_sparse	See the Python documentation of `is_sparse`.
validate_args	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.
mask	Optional logical vector or array masking observations (`FALSE` = ignored). Defaults to `py_none()`.
sample	Logical. If `TRUE`, directly draw values from the distribution and return them; if `FALSE` (default), create a random variable inside a model.
seed	Integer seed for this draw when `sample = TRUE`. By default (`py_none()`) the draw comes from the session's random stream, set by `importBF(rand_seed = ...)`, so consecutive draws differ but the whole sequence is reproducible. No effect when `sample = FALSE`.

shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF CAR distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the CAR distribution (for direct sampling).
- When create_obj = TRUE: The raw BF CAR object (for advanced use cases).

References

[1]: https://www.pymc.io/projects/examples/en/latest/spatial/conditional_autoregressive_priors.html "Conditional Autoregressive (CAR) Models for Spatial Data"

[2]: https://mc-stan.org/learn-stan/case-studies/icar_stan.html "Intrinsic Auto-Regressive Models for Areal Data - Stan"

[3]: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat08048 "Conditional Autoregressive (CAR) Model - Schmidt"

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.car(
    loc = 0.0,
    correlation = 0.9,
    conditional_precision = 1.0,
    adj_matrix = jnp$array(
        matrix(c(0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0), nrow = 3L, byrow = TRUE)
    ),
    sample = TRUE
)
```

`bf.dist.categorical` *Categorical distribution. distribution*

Description

The Categorical distribution, also known as the multinomial distribution, describes the probability of K different outcomes with probabilities $p_1, p_2, \ldots, p_K$.There's no inherent ordering required among the categories.

- The probabilities satisfy (p_i \ge 0) for all (i) and $\sum_{i=1}^{K} p_i = 1$.
- It's essentially the distribution of a single draw from a discrete set of categories.
- It generalises the Bernoulli distribution (which is the special case $K = 2$).

It is commonly used to model discrete choices or classifications.

If X is a categorical random variable taking values in $1, 2, \ldots, K$ with probabilities $p_1, \ldots, p_K$, then:

$$\Pr(X = i) = p_i, \quad i = 1, \ldots, K.$$

where p_k is the probability of outcome k, and the sum is over all possible outcomes.

Usage

```
bf.dist.categorical(
  probs = py_none(),
  logits = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  infer = py_none(),
  enum = FALSE,
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`probs`	A 1D array of probabilities for each category. Must sum to 1.
`logits`	A 1D array of unnormalized log probabilities for each category.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).

name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
infer	Optional named list passed to NumPyro's infer argument (e.g. list(enumerate = "parallel")).
enum	Logical. If TRUE, the discrete latent is summed out exactly (parallel enumeration) instead of sampled; same as infer = list(enumerate = "parallel"). Defaults to FALSE.
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Categorical distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Categorical distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

https://en.wikipedia.org/wiki/Categorical_distribution"Categorical distribution"

https://distribution-explorer.github.io/discrete/categorical.html"Categorical distribution — Probability Distribution Explorer documentation"

https://static.hlt.bme.hu/semantics/external/pages/n-gram/en.wikipedia.org/wiki/Categorical_distribution.html"Categorical distribution - Wikipedia"

See Also

https://num.pyro.ai/en/stable/distributions.html#categorical

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.categorical(probs = jnp$array(c(0.2, 0.3, 0.5)), sample = TRUE)
```

`bf.dist.cauchy` *Cauchy distribution*

Description

Samples from a Cauchy

The Cauchy distribution, also known as the Lorentz distribution, is a continuous probability distribution that arises frequently in various fields, including physics and statistics. It is characterized by its heavy tails, which extend indefinitely. This means it has a higher probability of extreme values compared to the normal distribution.

- **Parameters**:
- **Location parameter** (μ): Determines the peak of the distribution.
- **Scale parameter** (γ): Describes the half-width at half-maximum (HWHM) of the peak.

$$f(x; \mu, \gamma) = \frac{1}{\pi\gamma \left[1 + \left(\frac{x-\mu}{\gamma}\right)^2\right]}$$

where:

- x is the random variable,
- μ is the location parameter,
- γ is the scale parameter.

Usage

```
bf.dist.cauchy(
  loc = 0,
  scale = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
```

```
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	Location parameter. Defaults to 0.0.
scale	See the Python documentation of scale.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Cauchy distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Cauchy distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#cauchy

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.cauchy(loc = 0.0, scale = 1.0, sample = TRUE)
```

`bf.dist.chi2` *Chi-squared distribution*

Description

The Chi-squared distribution is a continuous probability distribution that arises frequently in hypothesis testing, particularly in ANOVA and chi-squared tests. It is defined by a single positive parameter, degrees of freedom (df), the number of independent standard normal variables squared and summed, which determines the shape of the distribution.

$$f(x;k) = \begin{cases} \frac{x^{\frac{k}{2}-1}e^{-x/2}}{2^{\frac{k}{2}}\Gamma\left(\frac{k}{2}\right)}, & x > 0 \\ 0, & \text{otherwise} \end{cases}$$

where:

- (x) is the random variable,
- (k) is the degrees of freedom,
- (\Gamma(\cdot)) is the gamma function.

Usage

```
bf.dist.chi2(
  df,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

df	Degrees of freedom. Must be positive.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Chi2 distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Chi2 distribution (for direct sampling).
- When create_obj = TRUE: The raw BF Chi2 object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#chi2

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.chi2(df = 3.0, sample = TRUE)
```

`bf.dist.delta` *Delta distribution*

Description

The Delta distribution, also known as a point mass distribution, assigns probability 1 to a single point and 0 elsewhere. It's useful for representing deterministic variables or as a building block for more complex distributions.

$$F(x) = \begin{cases} 0, & x < a, \\ 1, & x \geq a. \end{cases}$$

Usage

```
bf.dist.delta(
  v = 0,
  log_density = 0,
  event_dim = 0,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`v`	The location of the point mass.
`log_density`	The log probability density of the point mass. This is primarily for creating distributions that are non-normalized or for specific advanced use cases. For a standard delta distribution, this should be 0. Defaults to 0.0.
`event_dim`	The number of rightmost dimensions of `v` to interpret as event dimensions. Defaults to 0.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
`obs`	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.

mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Circulant Delta Distribution distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Delta distribution (for direct sampling).
- When create_obj = TRUE: The raw BF Circulant Delta Distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#delta

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.delta(v = 0.0, sample = TRUE)
```

`bf.dist.dirichlet` *Dirichlet distribution*

Description

Samples from a Dirichlet distribution.

The Dirichlet distribution is a multivariate generalization of the Beta distribution. It is a probability distribution over a simplex, which is a set of vectors where each element is non-negative and sums to one. It is often used as a prior distribution for categorical distributions.

For (X = (X_1, \dots, X_K) \sim \mathrm{Dir}(\alpha)), the density is

$$f(x;\alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^{K} x_i^{\alpha_i - 1}$$

for (x) in the simplex, and 0 otherwise. Here (B(\alpha)) is the multivariate Beta function (the normalization constant):

$$B(\alpha) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)}.$$

Usage

```
bf.dist.dirichlet(
  concentration,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`concentration`	The concentration parameter(s) of the Dirichlet distribution. Must be a positive array.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.

obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Dirichlet Distribution distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Dirichlet distribution (for direct sampling).
- When create_obj = TRUE: The raw BF Dirichlet Distribution object (for advanced use cases).

References

https://distribution-explorer.github.io/multivariate_continuous/dirichlet.html

https://en.wikipedia.org/wiki/Dirichlet_distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#dirichlet

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.dirichlet(concentration = jnp$array(c(1.0, 1.0, 1.0)), sample = TRUE)
```

`bf.dist.dirichlet_multinomial`
Dirichlet-Multinomial distribution

Description

Creates a Dirichlet-Multinomial compound distribution, which is a Multinomial distribution with a Dirichlet prior on its probabilities. It is often used in Bayesian statistics to model count data where the proportions of categories are uncertain.

The **Dirichlet-Multinomial** distribution describes a model in which you first draw a probability vector

$$\mathbf{p} = (p_1, \ldots, p_K) \sim \text{Dirichlet}(\alpha)$$

and then you draw counts

$$\mathbf{x} = (x_1, \ldots, x_K) \sim \text{Multinomial}(n, \mathbf{p}).$$

Integrating out $\mathbf{p}$ gives you the marginal distribution of $\mathbf{x}$.

- It generalizes the **Beta-Binomial** model to multiple categories (i.e. $K > 2$).
- A key feature is that it introduces **overdispersion** relative to a standard multinomial: counts are more "clumped" because the underlying probabilities vary according to the Dirichlet prior.

Usage

```
bf.dist.dirichlet_multinomial(
  concentration,
  total_count = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  infer = py_none(),
  enum = FALSE,
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

concentration	The concentration parameter (alpha) for the Dirichlet prior. Values must be positive. The last dimension is interpreted as the number of categories.
total_count	The total number of trials (n). This must be a non-negative integer. Defaults to 1.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
infer	Optional named list passed to NumPyro's infer argument (e.g. list(enumerate = "parallel")).
enum	Logical. If TRUE, the discrete latent is summed out exactly (parallel enumeration) instead of sampled; same as infer = list(enumerate = "parallel"). Defaults to FALSE.
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF dirichlet_multinomial Distribution distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the dirichlet_multinomial distribution (for direct sampling).
- When create_obj = TRUE: The raw BF dirichlet_multinomial Distribution object (for advanced use cases).

References

https://en.wikipedia.org/wiki/Dirichlet-multinomial_distribution

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.dirichlet_multinomial.html

See Also

https://num.pyro.ai/en/stable/distributions.html#dirichletmultinomial

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
counts <- bf.dist.dirichlet_multinomial(
    concentration = jnp$array(c(1.0, 1.0, 1.0)),
    total_count = 10L,
    sample = TRUE
)
```

---

bf.dist.discrete_uniform

Discrete Uniform distribution

Description

Samples from a Discrete Uniform distribution.

The Discrete Uniform distribution defines a uniform distribution over a range of integers. It is characterized by a lower bound (`low`) and an upper bound (`high`), inclusive.

$$P(X = k) = \frac{1}{high - low + 1}, \quad k \in \{low, low + 1, ..., high\}$$

Otherwise (if k is outside the range), $P(X = k) = 0$.

Usage

```
bf.dist.discrete_uniform(
  low = 0,
  high = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  infer = py_none(),
  enum = FALSE,
```

```
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

low	The lower bound of the uniform range, inclusive.
high	The upper bound of the uniform range, inclusive.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
infer	Optional named list passed to NumPyro's infer argument (e.g. list(enumerate = "parallel")).
enum	Logical. If TRUE, the discrete latent is summed out exactly (parallel enumeration) instead of sampled; same as infer = list(enumerate = "parallel"). Defaults to FALSE.
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF DiscreteUniform Distribution distribution object (for model building).

- When `sample = TRUE`: A JAX array of samples drawn from the DiscreteUniform distribution (for direct sampling).
- When `create_obj = TRUE`: The raw BF DiscreteUniform Distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#discreteuniform

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.discrete_uniform(low = 0, high = 5, sample = TRUE)
```

---

bf.dist.euler_maruyama

Euler–Maruyama distribution

Description

Euler–Maruyama methode is a method for the approximate numerical solution of a stochastic differential equation (SDE). It simulates the solution to an SDE by iteratively applying the Euler method to each time step, incorporating a random perturbation to account for the diffusion term.

$$dX_t = f(X_t, t)dt + g(X_t, t)dW_t$$

where:

- X_t is the state of the system at time t.
- $f(X_t, t)$ is the drift coefficient.
- $g(X_t, t)$ is the diffusion coefficient.
- dW_t is a Wiener process (Brownian motion).

Usage

```
bf.dist.euler_maruyama(
  t,
  sde_fn,
  init_dist,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
```

```
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

t	Discretized time steps.
sde_fn	A function that takes the current state and time as input and returns the drift and diffusion coefficients.
init_dist	The initial distribution of the system.
validate_args	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.
mask	Optional logical vector or array masking observations (`FALSE` = ignored). Defaults to `py_none()`.
sample	Logical. If `TRUE`, directly draw values from the distribution and return them; if `FALSE` (default), create a random variable inside a model.
seed	Integer seed for this draw when `sample = TRUE`. By default (`py_none()`) the draw comes from the session's random stream, set by `importBF(rand_seed = ...)`, so consecutive draws differ but the whole sequence is reproducible. No effect when `sample = FALSE`.
shape	Integer vector. With `sample = FALSE`, the batch shape of the distribution in the model; with `sample = TRUE`, the shape of the draw. Defaults to `c()` (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to `0`.
create_obj	Logical. If `TRUE`, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to `FALSE`.
to_jax	Logical. Return a JAX array (`TRUE`, default) or convert to R.

Value

- numeric vector or array: Samples drawn from the Euler–Maruyama distribution.

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.euler_maruyama(
    t = jnp$array(c(0.0, 0.1, 0.2)),
    sde_fn = function(x, t) reticulate::tuple(x, 1.0),
    init_dist = bf.dist.normal(0.0, 1.0, create_obj = TRUE),
    sample = TRUE
)
```

`bf.dist.exponential` *Exponential distribution*

Description

The Exponential distribution is a continuous probability distribution that models the time until an event occurs in a Poisson process, where events occur continuously and independently at a constant average rate. It is often used to model the duration of events, such as the time until a machine fails or the length of a phone call.

For (x \ge 0)

$$F(x; \lambda) = 1 - e^{-\lambda x}$$

Zhere (\lambda > 0) is the rate parameter (the mean is (1/\lambda)).

Usage

```
bf.dist.exponential(
  rate = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

rate	The rate parameter, λ. Must be positive.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Exponential distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Exponential distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

https://reference.wolfram.com/language/ref/ExponentialDistribution.html

https://en.wikipedia.org/wiki/Exponential_distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#exponential

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.exponential(rate = 1.0, sample = TRUE)
```

`bf.dist.gamma` *Gamma distribution*

Description

Samples from a Gamma distribution.

The Gamma distribution is a continuous probability distribution frequently used in Bayesian statistics, particularly as a prior distribution for variance parameters. It is defined by two positive shape parameters: concentration ((k)) and rate ((\lambda)).

There are two common parameterizations:

1. **Shape–scale**: parameters ((\alpha, \theta)), with density [f(x) = \frac{1}{\Gamma(\alpha) \theta^\alpha} x^{\alpha - 1} e^{-x/\theta}, \quad x \ge 0]
2. **Shape–rate**: parameters ((\alpha, \lambda)) where (\lambda = 1/\theta), with density [f(x) = \frac{\lambda^\alpha}{\Gamma(\alpha)} x^{\alpha - 1} e^{-\lambda x}, \quad x \ge 0]

Usage

```
bf.dist.gamma(
  concentration,
  rate = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`concentration`	The shape parameter of the Gamma distribution (k > 0).
`rate`	The rate parameter of the Gamma distribution (theta > 0).

validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- Gamma: A BF Gamma distribution object (for model building).
- numeric vector or array: A JAX array of samples drawn from the Gamma distribution (for direct sampling).
- Gamma: The raw BF distribution object (for advanced use cases).

References

https://reference.wolfram.com/language/ref/GammaDistribution.html

https://en.wikipedia.org/wiki/Gamma_distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#gamma

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.gamma(concentration = 2.0, rate = 0.5, sample = TRUE)
```

`bf.dist.gamma_poisson` *Gamma Poisson distribution*

Description

The Gamma-Poisson distribution, also known as the Negative Binomial distribution, models overdispersed count data. It arises from a hierarchical process where the rate parameter of a Poisson distribution is itself a random variable following a Gamma distribution. This structure allows the model to capture variability in count data that exceeds what is predicted by the Poisson distribution, making it suitable for applications like modeling RNA-sequencing data and microbial count.

$$f(x) = \frac{\Gamma(x+\beta)\alpha^x}{\Gamma(\beta)(1+\alpha)^{\beta+x}x!}, \quad x = 0, 1, 2, \ldots$$

Where Γ denotes the gamma function.

This CDF expresses the probability that the count variable (X) takes a value less than or equal to (x), incorporating the overdispersion introduced by the Gamma distribution.

Usage

```
bf.dist.gamma_poisson(
  concentration,
  rate = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`concentration`	Shape parameter (alpha) of the Gamma distribution.
`rate`	Rate parameter (beta) for the Gamma distribution.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
`obs`	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.

mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF GammaPoisson distribution object (for model
- building).
- When sample = TRUE: A JAX array of samples drawn from the GammaPoisson
- distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced
- use cases).

References

[1] https://www.statisticshowto.com/gamma-poisson-distribution

https://en.wikipedia.org/wiki/Negative_binomial_distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#gammapoisson

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.gamma_poisson(concentration = 1.0, rate = 2.0, sample = TRUE)
```

`bf.dist.gaussian_copula` *Gaussian Copula distribution*

Description

A distribution that links the `batch_shape\[:-1\]` of a marginal distribution with a multivariate Gaussian copula, modelling the correlation between the axes. A copula is a multivariate distribution over the uniform distribution on [0, 1]. The Gaussian copula links the marginal distributions through a multivariate normal distribution.

Usage

```
bf.dist.gaussian_copula(
  marginal_dist,
  correlation_matrix = py_none(),
  correlation_cholesky = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`marginal_dist`	Distribution whose last batch axis is to be coupled.
`correlation_matrix`	Correlation matrix of the coupling multivariate normal distribution. Defaults to py_none().
`correlation_cholesky`	Correlation Cholesky factor of the coupling multivariate normal distribution. Defaults to py_none().
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
`obs`	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.

mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF GaussianCopula distribution object: When sample = FALSE (for model building).
- JAX array: When sample = TRUE (for direct sampling).
- BF distribution object: When create_obj = TRUE (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#gaussiancopula

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.gaussian_copula(
    marginal_dist = bf.dist.beta(2.0, 5.0, create_obj = TRUE),
    correlation_matrix = jnp$array(
        matrix(c(1.0, 0.7, 0.7, 1.0), nrow = 2L, byrow = TRUE)
    ),
    sample = TRUE
)
```

`bf.dist.gaussian_copula_beta` *Gaussian Copula Beta distribution*

Description

This distribution combines a Gaussian copula with a Beta distribution. The Gaussian copula models the dependence structure between random variables, while the Beta distribution defines the marginal distributions of each variable.

Usage

```
bf.dist.gaussian_copula_beta(
  concentration1,
  concentration0,
  correlation_matrix = py_none(),
  correlation_cholesky = py_none(),
  validate_args = FALSE,
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`concentration1`	The first shape parameter of the Beta distribution.
`concentration0`	The second shape parameter of the Beta distribution.
`correlation_matrix`	Correlation matrix of the coupling multivariate normal distribution. Defaults to py_none().
`correlation_cholesky`	The Cholesky decomposition of the correlation matrix.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
`obs`	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.

mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- GaussianCopulaBeta: A BF GaussianCopulaBeta distribution object (for model building).
- numeric vector or array: A JAX array of samples drawn from the GaussianCopulaBeta distribution (for direct sampling).
- Distribution: The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#gaussiancopulabetadistribution

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.gaussian_copula_beta(
    concentration1 = jnp$array(c(2.0, 3.0)),
    concentration0 = jnp$array(c(5.0, 3.0)),
    correlation_cholesky = jnp$linalg$cholesky(
        jnp$array(matrix(c(1.0, 0.7, 0.7, 1.0), nrow = 2L, byrow = TRUE))
    ),
    sample = TRUE
)
```

`bf.dist.gaussian_random_walk` *Gaussian Random Walk distribution*

Description

Creates a distribution over a Gaussian random walk of a specified number of steps. This is a discrete-time stochastic process where the value at each step is the previous value plus a Gaussian-distributed increment. The distribution is over the entire path.

Usage

```
bf.dist.gaussian_random_walk(
  scale = 1,
  num_steps = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`scale`	The standard deviation (σ) of the Gaussian increments. Must be positive. Defaults to 1.0. - *num_steps (int, optional): The number of steps in the random walk, which determines the event shape of the distribution. Must be positive. Defaults to 1. - *validate_args (bool, optional): Whether to enable validation of distribution parameters. Defaults to `py_none()`.
`num_steps`	See the Python documentation of `num_steps`.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
`obs`	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.
`mask`	Optional logical vector or array masking observations (`FALSE` = ignored). Defaults to `py_none()`.
`sample`	Logical. If `TRUE`, directly draw values from the distribution and return them; if `FALSE` (default), create a random variable inside a model.

seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF.primitives.Messenger: A BF sample site object when used in a model context (sample = FALSE).
- numeric vector or array: A JAX array of samples drawn from the GaussianRandomWalk distribution (for direct sampling, sample = TRUE).
- numpyro.distributions.Distribution: The raw BF distribution object (if create_obj = TRUE).

See Also

https://num.pyro.ai/en/stable/distributions.html#gaussianrandomwalk

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
path <- bf.dist.gaussian_random_walk(scale = 0.5, num_steps = 100L, sample = TRUE)
```

---

bf.dist.gaussian_state_space

Gaussian State Space distribution

Description

Samples from a Gaussian state space model.

Usage

```
bf.dist.gaussian_state_space(
  num_steps,
  transition_matrix,
  covariance_matrix = py_none(),
  precision_matrix = py_none(),
  scale_tril = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

num_steps	Number of steps.
transition_matrix	State space transition matrix $\mathbf{A}$.
covariance_matrix	Covariance of the innovation noise ϵ. Defaults to py_none(). - *precision_matrix (numeric vector or array, optional): Precision matrix of the innovation noise ϵ. Defaults to py_none(). - *scale_tril (numeric vector or array, optional): Scale matrix of the innovation noise ϵ. Defaults to py_none().
precision_matrix	See the Python documentation of precision_matrix.
scale_tril	See the Python documentation of scale_tril.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed =

	...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF GaussianStateSpace distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the GaussianStateSpace distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#gaussianstate

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.gaussian_state_space(
    num_steps = 5L,
   transition_matrix = jnp$array(matrix(c(0.5, 0.1, 0.0, 0.8), nrow = 2L, byrow = TRUE)),
   covariance_matrix = jnp$array(matrix(c(1.0, 0.6, 0.6, 1.0), nrow = 2L, byrow = TRUE)),
    sample = TRUE
)
```

---

bf.dist.geometric *Geometric distribution. distribution*

Description

The Geometric distribution models the number of independent Bernoulli trials needed to obtain the first success (or equivalently the number of failures before the first success). Each trial has the same probability of success p, and trials are independent.

- X = number of trials until the first success (so support $1, 2, 3, \ldots$).

$$P(X = k) = (1 - p)^{\cdot k-1}; p, \quad k = 1, 2, \ldots$$

Usage

```
bf.dist.geometric(
  probs = py_none(),
  logits = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

probs	Probability of success on each trial. Must be between 0 and 1.
logits	Log-odds of success on each trial. probs = jax.nn.sigmoid(logits).
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When `sample = FALSE`: A BF Geometric distribution object (for model building).
- When `sample = TRUE`: A JAX array of samples drawn from the Geometric distribution (for direct sampling).
- When `create_obj = TRUE`: The raw BF distribution object (for advanced use cases).

References

https://www.britannica.com/topic/geometric-distribution "Geometric distribution | Definition, Formula, Examples, Illustration, & Applications | Britannica"

https://en.wikipedia.org/wiki/Geometric_distribution "Geometric distribution"

https://distribution-explorer.github.io/discrete/geometric.html "Geometric distribution — Probability Distribution Explorer documentation"

See Also

https://num.pyro.ai/en/stable/distributions.html#geometric

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.geometric(probs = 0.5, sample = TRUE)
```

`bf.dist.gompertz` *Gompertz distribution*

Description

The Gompertz distribution is a distribution with support on the positive real line that is closely related to the Gumbel distribution. This implementation follows the notation used in the Wikipedia entry for the Gompertz distribution. See https://en.wikipedia.org/wiki/Gompertz_distribution.

The probability density function (PDF) is:

$$F(x) = 1 - \exp!\big[-a(e^x - 1)\big], \quad x \geq 0.$$

Usage

```
bf.dist.gompertz(
  concentration,
  rate = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

concentration	The concentration parameter. Must be positive.
rate	The rate parameter. Must be positive.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF Gompertz distribution object: When `sample = FALSE` (for model building).
- JAX array: When `sample = TRUE` (for direct sampling).
- BF distribution object: When `create_obj = TRUE` (for advanced use cases).

References

https://en.wikipedia.org/wiki/Gompertz_distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#gompertz

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.gompertz(concentration = 1.0, rate = 1.0, sample = TRUE)
```

---

bf.dist.gumbel *Gumbel distribution*

Description

Samples from a Gumbel (or Extreme Value) distribution.

The Gumbel distribution is a continuous probability distribution named after German mathematician Carl Gumbel. It is often used to model the distribution of maximum values in a sequence of independent random variables.

$$F(x;\mu,\beta) = \exp\big(-\exp\big(-\tfrac{x-\mu}{\beta}\big)\big), \quad x \in \mathbb{R}.$$

Usage

```
bf.dist.gumbel(
  loc = 0,
  scale = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
```

```
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	Location parameter. Defaults to 0.0.
scale	Scale parameter. Must be positive. Defaults to 1.0.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Gumbel distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Gumbel distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

https://en.wikipedia.org/wiki/Gumbel_distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#gumbel

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.gumbel(loc = 0.0, scale = 1.0, sample = TRUE)
```

`bf.dist.half_cauchy` *HalfCauchy distribution*

Description

The HalfCauchy distribution is a probability distribution that is half of the Cauchy distribution. It is defined on the positive real numbers and is often used in situations where only positive values are relevant.

$$f(x) = \frac{1}{2} \cdot \frac{1}{\pi \cdot \frac{1}{scale} \cdot (x^2 + \frac{1}{scale^2})}$$

Usage

```
bf.dist.half_cauchy(
  scale = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`scale`	The scale parameter of the Cauchy distribution. Must be positive.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).

name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF HalfCauchy distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the HalfCauchy distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#halfcauchy

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.half_cauchy(scale = 1.0, sample = TRUE)
```

`bf.dist.half_normal` *HalfNormal distribution*

Description

Samples from a HalfNormal distribution.

The HalfNormal distribution is a distribution of the absolute value of a normal random variable. It is defined by a location parameter (implicitly 0) and a scale parameter.

$$f(x;\sigma) = \frac{\sqrt{2}}{\sigma\sqrt{\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right), \qquad x \geq 0.$$

Usage

```
bf.dist.half_normal(
  scale = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`scale`	The scale parameter of the distribution. Must be positive.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
`obs`	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.
`mask`	Optional logical vector or array masking observations (`FALSE` = ignored). Defaults to `py_none()`.
`sample`	Logical. If `TRUE`, directly draw values from the distribution and return them; if `FALSE` (default), create a random variable inside a model.
`seed`	Integer seed for this draw when `sample = TRUE`. By default (`py_none()`) the draw comes from the session's random stream, set by `importBF(rand_seed = ...)`, so consecutive draws differ but the whole sequence is reproducible. No effect when `sample = FALSE`.

shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF HalfNormal distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the HalfNormal distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

https://distribution-explorer.github.io/continuous/halfcauchy.html

See Also

https://num.pyro.ai/en/stable/distributions.html#halfnormal

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.half_normal(scale = 1.0, sample = TRUE)
```

---

bf.dist.inverse_gamma *Inverse Gamma distribution*

Description

The InverseGamma distribution is a two-parameter family of continuous probability distributions. It is defined by its shape α and rate β parameters. It is often used as a prior distribution for precision parameters (inverse variance) in Bayesian statistics.

$$f(x;\alpha,\beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{-(\alpha+1)} \exp\left(-\frac{\beta}{x}\right), \quad x > 0$$

where:

- (\Gamma(\alpha)) is the Gamma function.

Usage

```
bf.dist.inverse_gamma(
  concentration,
  rate = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

concentration	The concentration parameter α of the InverseGamma distribution. Must be positive.
rate	The rate parameter β of the InverseGamma distribution. Must be positive.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When `sample = FALSE`: A BF InverseGamma distribution object (for model building).
- When `sample = TRUE`: A JAX array of samples drawn from the InverseGamma distribution (for direct sampling).
- When `create_obj = TRUE`: The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#inversegamma

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.inverse_gamma(concentration = 2.0, rate = 1.0, sample = TRUE)
```

`bf.dist.kumaraswamy` *Kumaraswamy distribution*

Description

The Kumaraswamy distribution is a continuous probability distribution defined on the interval [0, 1]. It is a flexible distribution that can take on various shapes depending on its parameters.

f(x; a, b) = a b x^{a-1} (1 - x^a)^{b-1}, \quad x \in (0, 1) $$

where:

- (a) and (b) are shape parameters.
- (x) is the random variable.

Usage

```
bf.dist.kumaraswamy(
  concentration1,
  concentration0,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
```

```
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

concentration1	The first shape parameter. Must be positive.
concentration0	The second shape parameter. Must be positive.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Kumaraswamy distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Kumaraswamy distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#kumaraswamy

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.kumaraswamy(concentration1 = 2.0, concentration0 = 3.0, sample = TRUE)
```

`bf.dist.laplace` *Laplace distribution*

Description

Samples from a Laplace distribution, also known as the double exponential distribution. The Laplace distribution is defined by its location parameter (loc) and scale parameter (scale).

$$f(x \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$$

Where:

- (μ) is the **location parameter**, indicating the peak of the distribution.
- (b > 0) is the **scale parameter**, controlling the spread of the distribution.

Usage

```
bf.dist.laplace(
  loc = 0,
  scale = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`loc`	Location parameter of the Laplace distribution.
`scale`	See the Python documentation of `scale`.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).

name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Laplace distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Laplace distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#laplace

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.laplace(loc = 0.0, scale = 1.0, sample = TRUE)
```

`bf.dist.left_truncated_distribution`
Left Truncated distribution

Description

Samples from a left-truncated distribution.

A left-truncated distribution is a probability distribution obtained by restricting the support of another distribution to values greater than a specified lower bound. This is useful when dealing with data that is known to be greater than a certain value. All the "mass" below (or equal to) (a) is **excluded** (not just unobserved, but removed from the sample/analysis).

Let (X) be a random variable with PDF (f_X(x)) and CDF (F_X(x)). Choose a truncation point (a). Then define a new random variable (Y = X \mid (X > a)). The PDF and CDF of (Y) are:

- **PDF** for (y > a):

$$f_Y(y) = \frac{f_X(y)}{1 - F_X(a)}.$$

(Here (1 - F_X(a) = P(X > a)) is the normalising constant) ([Wikipédia][1])

- **CDF** for (y > a):

$$F_Y(y) = P(Y \leq y) = \frac{F_X(y) - F_X(a)}{1 - F_X(a)}.$$

Usage

```
bf.dist.left_truncated_distribution(
  base_dist,
  low = 0,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

base_dist	The base distribution to truncate. Must be univariate and have real support.
low	The lower truncation bound. Values less than this are excluded from the distribution.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF LeftTruncatedDistribution distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the LeftTruncatedDistribution distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

https://en.wikipedia.org/wiki/Truncated_distribution

https://encyclopediaofmath.org/wiki/Truncated_distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#lefttruncateddistribution

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.left_truncated_distribution(
    base_dist = bf.dist.normal(0.0, 1.0, create_obj = TRUE),
    low = 0.0,
    sample = TRUE
)
```

`bf.dist.levy` *Levy distribution*

Description

Samples from a Levy distribution. The Lévy distribution is a continuous probability distribution on the positive real line (or shifted positive line) that is *heavy-tailed*, *skewed*, and arises naturally in connection with stable distributions specifically the case with stability index $\alpha = \frac{1}{2}$. It is often used in contexts such as hitting-time problems for Brownian motion, physics (e.g., van der Waals line-shapes), and modelling very heavy-tailed phenomena. Let (X) be a Lévy-distributed random variable with **location** parameter μ and **scale** parameter (c > 0). The support is $x \geq \mu$.

PDF

$$f(x;\mu,c) = \sqrt{\frac{c}{2\pi}};\frac{1}{(x-\mu)^{3/2}};\exp!\Big(-\frac{c}{2(x-\mu)}\Big), \quad x \geq \mu.$$

CDF

$$F(x;\mu,c) = \Pr(X \leq x) = \operatorname{erfc}!\Big(\sqrt{\tfrac{c}{2(x-\mu)}}\Big) = 2 - 2,\Phi!\Big(\sqrt{\tfrac{c}{(x-\mu)}}\Big), \quad x \geq \mu.$$

Here erfc is the complementary error function and Φ is the standard normal CDF.

In the *standard* case with $\mu = 0$ and $c = 1$, one often simply writes $X \sim \text{Lévy}(0, 1)$.

Usage

```
bf.dist.levy(
  loc,
  scale,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
```

```
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	Location parameter.
scale	See the Python documentation of scale.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF Levy distribution object: When sample = FALSE (for model building).
- JAX array: When sample = TRUE (for direct sampling).
- BF distribution object: When create_obj = TRUE (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#levy

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.levy(loc = 0.0, scale = 1.0, sample = TRUE)
```

`bf.dist.lkj` *Lewandowski Kurowicka Joe (LKJ) distribution*

Description

The LKJ distribution is controlled by the concentration parameter η to make the probability of the correlation matrix M proportional to $\det(M)^{\eta-1}$. When $\eta = 1$, the distribution is uniform over correlation matrices. When $\eta > 1$, the distribution favors samples with large determinants. When $\eta < 1$, the distribution favors samples with small determinants.

$$P(M) \propto |\det(M)|^{\eta-1}$$

Usage

```
bf.dist.lkj(
  dimension,
  concentration = 1,
  sample_method = "onion",
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`dimension`	The dimension of the correlation matrices.
`concentration`	The concentration/shape parameter of the distribution (often referred to as eta). Must be positive.
`sample_method`	Either “cvine” or “onion”. Both methods are proposed in [1] and offer the same distribution over correlation matrices. But they are different in how to generate samples. Defaults to “onion”.

validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF LKJ distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the LKJ distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

https://en.wikipedia.org/wiki/Kumaraswamy_distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#lkj

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.lkj(dimension = 2L, concentration = 1.0, sample = TRUE)
```

`bf.dist.lkj_cholesky` *LKJ Cholesky distribution*

Description

The LKJ (Lewandowski-Kurowicka-Joe) Cholesky distribution is a family of distributions on symmetric matrices, often used as a prior for the Cholesky decomposition of a symmetric matrix. It is particularly useful in BayesForge for models with covariance structure. Given a lower triangular matrix (L) with unit diagonal entries, the PDF is:

$$p(L|\eta) \propto \prod_{k=2}^{d} L_{kk}^{d-k+2\eta-2}$$

where:

- (L_{kk}) is the diagonal element of (L).
- (\eta) is the shape parameter.
- (d) is the dimension of the matrix.
- (\eta = 1): Uniform prior over correlation matrices.
- (\eta > 1): Prior favors correlation matrices close to the identity matrix (i.e., weak correlations).
- (\eta < 1): Prior allows stronger correlations.

Usage

```
bf.dist.lkj_cholesky(
  dimension,
  concentration = 1,
  sample_method = "onion",
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

dimension	See the Python documentation of dimension.
concentration	See the Python documentation of concentration.
sample_method	See the Python documentation of sample_method.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

A random variable (model building) or draws (sample = TRUE).

References

https://mc-stan.org/docs/2_21/functions-reference/cholesky-lkj-correlation-distribution.html

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.lkj_cholesky(dimension = 3L, concentration = 2.0, sample = TRUE)
```

`bf.dist.logistic` *Logistic distribution*

Description

Samples from a Logistic distribution.

The Logistic distribution is a continuous probability distribution defined by two parameters: location and scale. It is often used to model growth processes and is closely related to the normal distribution.Its CDF is the logistic (sigmoid) function, which makes it appealing in modeling probabilities, logistic regression, and various growth models. It resembles the normal distribution in shape (bell-shaped, symmetric) but has **heavier tails** (i.e. more probability in the extremes) and simpler closed-form expressions for the CDF.

- **Location parameter**: $\mu \in \mathbb{R}$\ — the center or "mean" of the distribution.
- **Scale parameter**: $s > 0$\ — controls the spread (similar role to standard deviation).
- **Support**: $x \in (-\infty, +\infty)$\.

Let (X \sim \mathrm{Logistic}(\mu, s)). Then:

$$F(x; \mu, s) = \frac{1}{1 + \exp\left(-\frac{x-\mu}{s}\right)}.$$

Equivalently:

$$F(x) = \frac{1}{2} + \frac{1}{2}\,\tanh!\left(\frac{x-\mu}{2s}\right).$$

Usage

```
bf.dist.logistic(
  loc = 0,
  scale = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	The location parameter, specifying the median of the distribution. Defaults to 0.0.
scale	The scale parameter, which determines the spread of the distribution. Must be positive. Defaults to 1.0.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF Logistic distribution object (for model building) when sample = FALSE.
- JAX array of samples drawn from the Logistic distribution (for direct sampling) when sample = TRUE.
- The raw BF distribution object (for advanced use cases) when create_obj = TRUE.

References

https://en.wikipedia.org/wiki/Logistic_distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#logistic

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.logistic(loc = 0.0, scale = 1.0, sample = TRUE)
```

`bf.dist.log_normal` *Log-Normal distribution*

Description

The Log-Normal distribution is a probability distribution defined for positive real-valued random variables, parameterized by a location parameter (loc : μ) and a scale parameter (scale). It arises when the logarithm of a random variable is normally distributed.

- A random variable (X) is **log-normal** if its natural logarithm (\ln X) is (approximately) normally distributed. Equivalently, one can write:

$$X = \exp(Y), \quad \text{where } Y \sim \mathcal{N}(\mu, \sigma^2).$$

Because (X) is the exponential of a normal, it is always positive (support (x>0)). It is useful in contexts where growth, multiplicative effects, or compounding factors dominate (e.g. stock prices, income, sizes of biological organisms).

Usage

```
bf.dist.log_normal(
  loc = 0,
  scale = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	Location parameter.
scale	Scale parameter.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF LogNormal distribution object (for model building).
- JAX array of samples drawn from the LogNormal distribution (for direct sampling).
- The raw BF distribution object (for advanced use cases).

References

https://en.wikipedia.org/wiki/Log-normal_distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#lognormal

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.log_normal(loc = 0.0, scale = 1.0, sample = TRUE)
```

`bf.dist.log_uniform` *Log-Uniform distribution*

Description

Samples from a Log Uniform distribution.

The Log Uniform distribution is defined over the positive real numbers and is the result of applying an exponential transformation to a uniform distribution over the interval [low, high]. It is often used when modeling parameters that must be positive.

A random variable X is **log-uniform** on $[a, b]$, with $0 < a < b$, if $\ln X$ is uniformly distributed on $[\ln a, \ln b]$. Equivalently, the density of X is proportional to $1/x$ over that interval. This distribution is sometimes called the *reciprocal distribution*. It is useful in modeling scales spanning several orders of magnitude, where you want every decade (or log-interval) to have equal weight.

For $x \in [a, b]$:

- **PDF**

$$f(x) = \frac{1}{x, [\ln(b) - \ln(a)]}, \quad a \le x \le b.$$

Outside $[a, b]$, $f(x) = 0$.

- **CDF**

$$F(x) = \Pr(X \le x) = \frac{\ln(x) - \ln(a)}{\ln(b) - \ln(a)}, \quad a \le x \le b.$$

(For $x < a$, $F(x) = 0$; for $x > b$, $F(x) = 1$.

Usage

```
bf.dist.log_uniform(
  low,
  high,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
```

```
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

low	The lower bound of the uniform distribution's log-space. Must be positive.
high	The upper bound of the uniform distribution's log-space. Must be positive.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF LogUniform distribution object (for model building) when sample = FALSE.
- JAX array of samples drawn from the LogUniform distribution (for direct sampling) when sample = TRUE.
- The raw BF distribution object (for advanced use cases) when create_obj = TRUE.

References

https://en.wikipedia.org/wiki/Reciprocal_distribution

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.loguniform.html

https://docs.scipy.org/doc/scipy/tutorial/stats/continuous_loguniform.html

See Also

https://num.pyro.ai/en/stable/distributions.html#loguniform

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.log_uniform(low = 0.1, high = 10.0, sample = TRUE)
```

---

bf.dist.lower_truncated_power_law

Lower Truncated Power Law distribution

Description

The *Lower-Truncated Power-Law* distribution (also known as the *Pareto Type I* or *power-law with a lower bound*) models quantities that follow a heavy-tailed power-law behavior but are bounded below by a minimum value (x_{\min}). It is commonly used to describe phenomena such as wealth distributions, city sizes, and biological scaling laws.

The probability density function (PDF) is given by:

For (x \ge x_{\min} > 0):

$$f(x \mid \alpha, x_{\min}) = \frac{\alpha - 1}{x_{\min}} \left(\frac{x_{\min}}{x} \right)^{\alpha}$$

where $\alpha > 1$ is the shape (scaling) parameter.

Usage

```
bf.dist.lower_truncated_power_law(
  alpha,
  low,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
```

```
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

alpha	See the Python documentation of alpha.
low	See the Python documentation of low.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF LowerTruncatedPowerLaw distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the LowerTruncatedPowerLaw distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

Power-law distribution – Wikipedia

Pareto distribution – Wikipedia

Truncated Pareto distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#lowertruncatedpowerlaw

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.lower_truncated_power_law(alpha = -2.0, low = 1.0, sample = TRUE)
```

---

bf.dist.low_rank_multivariate_normal

Low Rank Multivariate Normal distribution

Description

The *Low-Rank Multivariate Normal* (LRMVN) distribution is a parameterization of the multivariate normal distribution where the covariance matrix is expressed as a low-rank plus diagonal decomposition:

$$\Sigma = FF^\top + D$$

where F is a low-rank matrix (capturing correlations) and D is a diagonal matrix (capturing independent noise). This representation is often used in probabilistic modeling and variational inference to efficiently handle high-dimensional Gaussian distributions with structured covariance.

For a (d)-dimensional random vector (x \in \mathbb{R}^d),

$$p(x \mid \mu, F, D) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right),$$

where (\Sigma = F F^\top + D).

Usage

```
bf.dist.low_rank_multivariate_normal(
  loc,
  cov_factor,
  cov_diag,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	Mean vector. cov_factor (numeric vector or array): Matrix used to construct the covariance matrix. cov_diag (numeric vector or array): Diagonal elements of the covariance matrix.
cov_factor	See the Python documentation of cov_factor.
cov_diag	See the Python documentation of cov_diag.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.

`create_obj`	Logical. If `TRUE`, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to `FALSE`.
`to_jax`	Logical. Return a JAX array (`TRUE`, default) or convert to R.

Value

A random variable (model building) or draws (`sample = TRUE`).

References

Multivariate normal distribution – Wikipedia

TensorFlow Probability: LowRankMultivariateNormal

See Also

`https://num.pyro.ai/en/stable/distributions.html#lowrankmultivariatenormal`

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
event_size <- 100L
rank <- 5L
bf.dist.low_rank_multivariate_normal(
    loc = bf.dist.normal(0, 1, shape = c(event_size), sample = TRUE) * 2,
    cov_factor = bf.dist.normal(0, 1, shape = c(event_size, rank), sample = TRUE),
    cov_diag = jnp$exp(bf.dist.normal(0, 1, shape = c(event_size), sample = TRUE)) * 0.1,
    sample = TRUE
)
```

---

bf.dist.matrix_normal *Matrix Normal distribution*

Description

Samples from a Matrix Normal distribution, which is a multivariate normal distribution over matrices. The distribution is characterized by a location matrix and two lower triangular matrices that define the correlation structure. The distribution is related to the multivariate normal distribution in the following way. If X $MN(loc, U, V)$ then $vec(X)$ $MVN(vec(loc), kron(V, U))$.

Let (X) be (n \times p), with mean matrix

$$M \in \mathbb{R}^{n \times p}$$

, row-covariance matrix $U \in \mathbb{R}^{n \times n}$ (positive-definite), and column-covariance matrix $V \in \mathbb{R}^{p \times p}$(positive-definite). Then the PDF is:

$$f(X; M, U, V) = \frac{1}{(2\pi)^{\frac{np}{2}}, |V|^{\frac{n}{2}}, |U|^{\frac{p}{2}}}, \exp!\Big[-\tfrac{1}{2}, \mathrm{tr}!\big(V^{-1}(X-M)^{\top}, U^{-1}(X-M)\big)\Big].$$

Also the equivalence to vec-form is given:

$$\mathrm{vec}(X) \sim \mathcal{N}_{np}\big(\mathrm{vec}(M), , V \otimes U\big).; ; \text{([The Book of Statistical Proofs][3])}$$

Usage

```
bf.dist.matrix_normal(
  loc,
  scale_tril_row,
  scale_tril_column,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`loc`	See the Python documentation of `loc`.
`scale_tril_row`	See the Python documentation of `scale_tril_row`.
`scale_tril_column`	See the Python documentation of `scale_tril_column`.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
`obs`	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.
`mask`	Optional logical vector or array masking observations (`FALSE` = ignored). Defaults to `py_none()`.
`sample`	Logical. If `TRUE`, directly draw values from the distribution and return them; if `FALSE` (default), create a random variable inside a model.
`seed`	Integer seed for this draw when `sample = TRUE`. By default (`py_none()`) the draw comes from the session's random stream, set by `importBF(rand_seed = ...)`, so consecutive draws differ but the whole sequence is reproducible. No effect when `sample = FALSE`.

shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF MatrixNormal distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the MatrixNormal distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

[1]: https://en.wikipedia.org/wiki/Matrix_normal_distribution "Matrix normal distribution"

[2]: https://statproofbook.github.io/P/matn-pdf.html "Probability density function of the matrix-normal distribution | The Book of Statistical Proofs"

See Also

https://num.pyro.ai/en/stable/distributions.html#matrixnormal_lowercase

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
n_rows <- 3L
n_cols <- 4L
loc <- jnp$zeros(c(n_rows, n_cols))
U_row_cov <- jnp$array(
    matrix(c(1.0, 0.5, 0.2, 0.5, 1.0, 0.3, 0.2, 0.3, 1.0), nrow = 3L, byrow = TRUE)
)
V_col_cov <- jnp$array(
    matrix(
        c(
           2.0, -0.8, 0.1, 0.4, -0.8, 2.0, 0.2, -0.2, 0.1, 0.2, 2.0, 0.0, 0.4, -0.2, 0.0,
            2.0
        ),
        nrow = 4L,
        byrow = TRUE
    )
)
bf.dist.matrix_normal(
    loc = loc,
```

```
    scale_tril_row = jnp$linalg$cholesky(U_row_cov),
    scale_tril_column = jnp$linalg$cholesky(V_col_cov),
    sample = TRUE
)
```

`bf.dist.mixture` *A marginalized finite mixture of component distributions. distribution*

Description

A **marginalised finite mixture of component distributions** refers to a probability model in which you have a finite number K of component distributions (e.g., normals, Poissons, etc.), each weighted by a mixing probability from a Categorical distribution. We **marginalise out** the latent assignment variable (i.e., you don't explicitly model which component each data point came from). The resulting distribution can be either a `mixture_general` (when component distributions are a list) or a `mixture_same_Family` (when component distributions are a single distribution).

Suppose you have:

- Mixing weights $\pi_1, \ldots, \pi_K$, with $\pi_j \geq 0$ and $\sum_{j=1}^{K} \pi_j = 1$.
- Component distributions (densities or PMFs) $f_j(x \mid \theta_j)$, for $j = 1, \ldots, K$.

Then the marginalised mixture distribution for an observed X is:

$$p(X = x) = \sum_{j=1}^{K} \pi_j; f_j(x \mid \theta_j).$$

This is exactly the mixture density/PMF you get when you marginalise the latent component-assignment variable $Z \in 1, \ldots, K$ out of the joint distribution:

$$p(x) = \sum_{z=1}^{K} p(z = j); p(x \mid z = j) = \sum_{j=1}^{K} \pi_j; f_j(x \mid \theta_j).$$

Usage

```
bf.dist.mixture(
  mixing_distribution,
  component_distributions,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
```

```
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

mixing_distribution	The distribution used to determine the mixing weights.
component_distributions	The list of component distributions.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Mixture distribution object (for model
- building).
- When sample = TRUE: A JAX array of samples drawn from the Mixture
- distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced
- use cases).

References

https://en.wikipedia.org/wiki/Mixture_distribution "Mixture distribution"

https://pymc3-testing.readthedocs.io/en/rtd-docs/notebooks/marginalized_gaussian_mixture_model.html "Marginalized Gaussian Mixture Model — PyMC3 3.1rc3 documentation"

See Also

https://num.pyro.ai/en/stable/distributions.html#mixture

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.mixture(
    mixing_distribution = bf.dist.categorical(
        probs = jnp$array(c(0.3, 0.7)),
        create_obj = TRUE
    ),
    component_distributions = list(
        bf.dist.normal(0, 1, create_obj = TRUE),
        bf.dist.normal(2, 1, create_obj = TRUE)
    ),
    sample = TRUE,
    shape = c(10L)
)
```

`bf.dist.mixture_general`

Mixture General distribution

Description

A mixture distribution is a probability distribution constructed by selecting one of several component distributions according to specified weights, and then drawing a sample from the chosen component. It allows modelling of heterogeneous populations and multimodal data.

Let the mixture have (K) component distributions with densities (f_i(x)) for (i = 1,\dots,K). Let weights (w_i\ge0) satisfy (\sum_{i=1}^K w_i = 1). Then the PDF is

$$f(x) = \sum_{i=1}^{K} w_i; f_i(x).$$

Parameters:

- **mixing_distribution**: A `Categorical` distribution representing the mixing weights.

- **component_distributions**: A list of distributions representing the components of the mixture.
- **sample (bool, optional): A control-flow argument. If `TRUE`, the function will directly sample a raw JAX array from the distribution, bypassing the BF model context. If `FALSE`, it will create a `BF.sample` site within a model. Defaults to `FALSE`.
- `shape` (integer vector): A multi-purpose argument for shaping. When `sample = FALSE` (model building), this is used with `shape` to set the distribution's batch shape. When `sample = TRUE` (direct sampling), this is used as `sample_shape` to draw a raw JAX array of the given shape.
- **seed (int, optional)**: An integer seed for this draw when `sample = TRUE`. By default (`py_none()`) the draw comes from the session's random stream, set by `bf(rand_seed=...)`, so consecutive draws differ but the whole sequence is reproducible. No effect when `sample = FALSE`.
- **obs (numeric vector or array, optional)**: The observed value for this random variable. If provided, the sample site is conditioned on this value, and the function returns the observed value. If `py_none()`, the site is treated as a latent (unobserved) random variable. Defaults to `py_none()`.
- **name (str, optional)**: The name of the sample site in a BF model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- **When** `sample = FALSE`: A BF MixtureGeneral distribution object (for model building).
- **When** `sample = TRUE`: A JAX array of samples drawn from the MixtureGeneral distribution (for direct sampling).
- **When** `create_obj = TRUE`: The raw BF distribution object (for advanced use cases).

Example Usage:

from BayesForge import bf m = bf(platform='cpu') bf.dist.mixture_general(mixing_distribution=bf.dist.categorical(probs=j
0.7]), create_obj = TRUE), component_distributions=[bf.dist.normal(loc=0.0, scale=1.0, create_obj
= TRUE),bf.dist.normal (loc=0.0, scale=1.0, create_obj = TRUE)], sample = TRUE)

Usage

```
bf.dist.mixture_general(
  mixing_distribution,
  component_distributions,
  support = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

mixing_distribution	See the Python documentation of mixing_distribution.
component_distributions	See the Python documentation of component_distributions.
support	See the Python documentation of support.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

A random variable (model building) or draws (sample = TRUE).

References

[1]: https://preliz.readthedocs.io/en/latest/distributions/gallery/mixture.html "Mixture Distribution — PreliZ 0.22.0 documentation"

[2]: https://reliability.readthedocs.io/en/latest/Mixture\ "Mixture models — reliability 0.9.0 documentation"

[3]: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.Mixture.html "Mixture — SciPy v1.16.2 Manual"

See Also

https://num.pyro.ai/en/stable/distributions.html#mixturegeneral

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.mixture_general(
    mixing_distribution = bf.dist.categorical(
        probs = jnp$array(c(0.3, 0.7)),
        create_obj = TRUE
    ),
    component_distributions = list(
        bf.dist.normal(-2.0, 1.0, create_obj = TRUE),
        bf.dist.normal(2.0, 1.0, create_obj = TRUE)
    ),
    sample = TRUE
)
```

---

bf.dist.mixture_same_family

Finite mixture of component distributions from the same family. distribution

---

Description

A *Mixture (Same-Family)* distribution is a finite mixture in which **all components come from the same parametric family** (for example, all Normal distributions but with different parameters), and are combined via mixing weights. The class is typically denoted as:

Let the mixture have (K) components. Let weights (w_i\ge0), (\sum_{i=1}^K w_i = 1). Let the component family have density (f(x \mid \theta_i)) for each component (i). Then the mixture's PDF is

$$f_X(x) = \sum_{i=1}^{K} w_i; f(x \mid \theta_i).$$

where each $f(x \mid \theta_i)$ is from the same family with parameter θ_i.

Usage

```
bf.dist.mixture_same_family(
  mixing_distribution,
  component_distribution,
  validate_args = py_none(),
```

```
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`mixing_distribution`	A `Categorical` distribution representing the mixing weights.
`component_distribution`	See the Python documentation of `component_distribution`.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
`obs`	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.
`mask`	Optional logical vector or array masking observations (`FALSE` = ignored). Defaults to `py_none()`.
`sample`	Logical. If `TRUE`, directly draw values from the distribution and return them; if `FALSE` (default), create a random variable inside a model.
`seed`	Integer seed for this draw when `sample = TRUE`. By default (`py_none()`) the draw comes from the session's random stream, set by `importBF(rand_seed = ...)`, so consecutive draws differ but the whole sequence is reproducible. No effect when `sample = FALSE`.
`shape`	Integer vector. With `sample = FALSE`, the batch shape of the distribution in the model; with `sample = TRUE`, the shape of the draw. Defaults to `c()` (no extra dimensions).
`event`	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to `0`.
`create_obj`	Logical. If `TRUE`, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to `FALSE`.
`to_jax`	Logical. Return a JAX array (`TRUE`, default) or convert to R.

Value

- **When** `sample = FALSE`: A BF MixtureSameFamily distribution object (for model building).
- **When** `sample = TRUE`: A JAX array of samples drawn from the MixtureSameFamily distribution (for direct sampling).
- **When** `create_obj = TRUE`: The raw BF distribution object (for advanced use cases).

References

Mixture distributions

See Also

https://num.pyro.ai/en/stable/distributions.html#mixture-,

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.mixture_same_family(
    mixing_distribution = bf.dist.categorical(
        probs = jnp$array(c(0.3, 0.7)),
        create_obj = TRUE
    ),
    component_distribution = bf.dist.normal(
        loc = 0.0,
        scale = 1.0,
        shape = c(2L),
        create_obj = TRUE
    ),
    sample = TRUE
)
```

`bf.dist.multinomial` *Multinomial distribution*

Description

The multinomial distribution is a discrete probability distribution that generalizes the Binomial distribution to the case of **more than two possible outcomes per trial**. Specifically:

- You perform a fixed number n of independent trials.
- On each trial there are k possible outcomes, labeled (1,2,\dots,k).
- The probability of outcome i on any trial is (p_i), where (p_1 + p_2 + \cdots + p_k = 1).
- Let X_i be the number of trials (out of n) that produced outcome i. Then the vector $\mathbf{X} = (X_1, \ldots, X_k)$ follows a multinomial distribution.

It reduces to the binomial when (k=2).

PMF (Probability Mass Function):

For $\mathbf{x} = (x_1, \ldots, x_k)$ with each $x_i \geq 0$ integer and $\sum_{i=1}^{k} x_i = n$,

$$\Pr(X_1 = x_1, \ldots, X_k = x_k) = \frac{n!}{x_1!, x_2! \cdots x_k!}; p_1^{x_1}; p_2^{x_2} \cdots p_k^{x_k}.$$

If $\sum_i x_i \neq n$, the probability is 0.

Usage

```
bf.dist.multinomial(
  total_count = 1,
  probs = py_none(),
  logits = py_none(),
  total_count_max = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  infer = py_none(),
  enum = FALSE,
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

total_count	The number of trials.
probs	Event probabilities. Must sum to 1.
logits	Event log probabilities. - *total_count_max (int, optional): An optional integer providing an upper bound on total_count. This is used for performance optimization with lax.scan when total_count is a dynamic JAX tracer, helping to avoid recompilation.
total_count_max	See the Python documentation of total_count_max.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
infer	Optional named list passed to NumPyro's infer argument (e.g. list(enumerate = "parallel")).
enum	Logical. If TRUE, the discrete latent is summed out exactly (parallel enumeration) instead of sampled; same as infer = list(enumerate = "parallel"). Defaults to FALSE.
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.

seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Multinomial distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Multinomial distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

https://online.stat.psu.edu/stat504/book/export/html/667 "2.3 - The Multinomial Distribution - STAT ONLINE"

https://en.wikipedia.org/wiki/Multinomial_distribution "Multinomial distribution"

https://faculty.washington.edu/yenchic/20A_stat512/Lec7_Multinomial.pdf "[PDF] Lecture 7: Multinomial distribution"

See Also

https://num.pyro.ai/en/stable/distributions.html#multinomial

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.multinomial(total_count = 10L, probs = jnp$array(c(0.2, 0.3, 0.5)), sample = TRUE)
```

`bf.dist.multinomial_logits`
Multinomial Logits distribution

Description

A *multinomial logits* distribution refers to a categorical (or more generally multinomial) distribution over (K) classes whose probabilities are given via the softmax of a vector of logits. That is, given a vector of real-valued logits $\ell = (\ell_1, \ldots, \ell_K)$, the class probabilities are:

$$p_k = \frac{\exp(\ell_k)}{\sum_{j=1}^{K} \exp(\ell_j)}.$$

Then a single draw from the distribution yields one of the (K) classes (or for a multinomial count version, counts over the classes) with those probabilities.

Categorical version (one draw): Let (X \in {1,2,\dots,K}) be the class. Then

$$\Pr(X = k \mid \ell); =; p_k = \frac{\exp(\ell_k)}{\sum_{j=1}^{K} \exp(\ell_j)}.$$

This directly uses the logits ℓ.

Multinomial version (n draws): If you draw (n) independent draws (or equivalently count vector) with probabilities (p = (p_1,. . . ,p_K)), then for a count vector (x = (x_1,. . . ,x_K)) with (\sum_k x_k = n),

$$\Pr(X = x \mid n, p) = \frac{n!}{x_1! \cdots x_K!}; \prod_{k=1}^{K} p_k^{x_k}$$

and p_k is given by the softmax of logits.

Usage

```
bf.dist.multinomial_logits(
  logits,
  total_count = 1,
  total_count_max = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  infer = py_none(),
  enum = FALSE,
  sample = FALSE,
  seed = py_none(),
  shape = c(),
```

```
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

logits	Logits for each outcome. Must be at least one-dimensional.
total_count total_count_max	The total number of trials. See the Python documentation of total_count_max.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
infer	Optional named list passed to NumPyro's infer argument (e.g. list(enumerate = "parallel")).
enum	Logical. If TRUE, the discrete latent is summed out exactly (parallel enumeration) instead of sampled; same as infer = list(enumerate = "parallel"). Defaults to FALSE.
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF MultinomialLogits distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the MultinomialLogits distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

https://en.wikipedia.org/wiki/Multinomial_distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#multinomiallogits

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.multinomial_logits(
    logits = jnp$array(c(1.0, 0.5)),
    total_count = 5L,
    sample = TRUE
)
```

`bf.dist.multinomial_probs`

Multinomial Probs distribution

Description

Samples from a Multinomial distribution.

The Multinomial distribution models the number of times each of several discrete outcomes occurs in a fixed number of trials. Each trial independently results in one of several outcomes, and each outcome has a probability of occurring.

Usage

```
bf.dist.multinomial_probs(
  probs,
  total_count = 1,
  total_count_max = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  infer = py_none(),
  enum = FALSE,
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
```

```
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

probs	Vector of probabilities for each outcome. Must sum to 1.
total_count total_count_max	The number of trials. See the Python documentation of total_count_max.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
infer	Optional named list passed to NumPyro's infer argument (e.g. list(enumerate = "parallel")).
enum	Logical. If TRUE, the discrete latent is summed out exactly (parallel enumeration) instead of sampled; same as infer = list(enumerate = "parallel"). Defaults to FALSE.
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF MultinomialProbs distribution object (for model building).
- JAX array of samples drawn from the MultinomialProbs distribution (for direct sampling).
- The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#multinomialprobs

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.multinomial_probs(
    probs = jnp$array(c(0.2, 0.3, 0.5)),
    total_count = 10L,
    sample = TRUE
)
```

`bf.dist.multivariate_normal`

Multivariate Normal distribution

Description

The Multivariate Normal distribution, also known as the Gaussian distribution in multiple dimensions, is a probability distribution that arises frequently in statistics and machine learning. It is defined by its mean vector and covariance matrix, which describe the central tendency and spread of the distribution, respectively.

$$p(x) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

where:

- x is a n-dimensional vector of random variables.
- μ is the mean vector.
- Σ is the covariance matrix.

Usage

```
bf.dist.multivariate_normal(
  loc = 0,
  covariance_matrix = py_none(),
  precision_matrix = py_none(),
  scale_tril = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
```

```
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	The mean vector of the distribution.
covariance_matrix	The covariance matrix of the distribution. Must be positive definite.
precision_matrix	The precision matrix (inverse of the covariance matrix) of the distribution. Must be positive definite.
scale_tril	The lower triangular Cholesky decomposition of the covariance matrix. - shape (integer vector): A multi-purpose argument for shaping. When sample = FALSE (model building), this is used with shape to set the distribution's batch shape. When sample = TRUE (direct sampling), this is used as sample_shape to draw a raw JAX array of the given shape.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When `sample = FALSE`: A BF MultivariateNormal distribution object (for model building).
- When `sample = TRUE`: A JAX array of samples drawn from the MultivariateNormal distribution (for direct sampling).
- When `create_obj = TRUE`: The raw BF distribution object (for advanced use cases).

References

Multivariate normal distribution – Wikipedia

See Also

https://num.pyro.ai/en/stable/distributions.html#multivariate-normal

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.multivariate_normal(
    loc = jnp$array(c(1.0, 0.0, -2.0)),
    covariance_matrix = jnp$array(
        matrix(c(2.0, 0.7, -0.3, 0.7, 1.0, 0.5, -0.3, 0.5, 1.5), nrow = 3L, byrow = TRUE)
    ),
    sample = TRUE
)
```

---

bf.dist.multivariate_student_t

Multivariate Student's t distribution

Description

The Multivariate Student's t distribution is a generalization of the Student's t distribution to multiple dimensions. It is a heavy-tailed distribution that is often used to model data that is not normally distributed.

The PDF of the multivariate Student's t-distribution for a random vector (\mathbf{x} \in \mathbb{R}^d) is given by:

$$f(\mathbf{x}) = \frac{\Gamma\left(\frac{\nu+d}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)\nu^{d/2}\pi^{d/2}|\Sigma|^{1/2}}\left(1+\frac{1}{\nu}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)^{-(\nu+d)/2}$$

where:

- $\Gamma(\cdot)$ is the Gamma function.

- μ is the mean vector.
- Σ is the scale (covariance) matrix.
- ν is the degrees of freedom.
- $d$is the dimensionality of$\mathbf{x}$.

Usage

```
bf.dist.multivariate_student_t(
  df,
  loc = 0,
  scale_tril = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

df	Degrees of freedom, must be positive.
loc	Location vector, representing the mean of the distribution. scale_tril (numeric vector or array): Lower triangular matrix defining the scale. - shape (integer vector): A multi-purpose argument for shaping. When sample = FALSE (model building), this is used with shape to set the distribution's batch shape. When sample = TRUE (direct sampling), this is used as sample_shape to draw a raw JAX array of the given shape.
scale_tril	See the Python documentation of scale_tril.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.

seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF MultivariateStudentT distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the MultivariateStudentT distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

Multivariate Student-t distribution – Wikipedia

See Also

https://num.pyro.ai/en/stable/distributions.html#multivariatestudentt

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.multivariate_student_t(
    df = 2,
    loc = jnp$array(c(1.0, 0.0, -2.0)),
    scale_tril = jnp$linalg$cholesky(
        jnp$array(
            matrix(
                c(2.0, 0.7, -0.3, 0.7, 1.0, 0.5, -0.3, 0.5, 1.5),
                nrow = 3L,
                byrow = TRUE
            )
        )
    ),
    sample = TRUE
)
```

`bf.dist.negative_binomial` *Negative Binomial distribution*

Description

The Negative Binomial distribution models the number of failures (or the total number of trials) in a sequence of independent Bernoulli trials with success probability (p), until a specified number `total_count` (r) of successes is achieved. It is often used as a count-data model when the variance exceeds the mean ("overdispersion") relative to a Poisson.

PMF (Probability Mass Function): Given (X) = number of failures before the (r)th success (so (X = 0,1,2,\dots)), with (r) successes fixed, success probability (p), failure probability (q = 1 - p):

$$\Pr(X = k) = \binom{k + r - 1}{k} p^r q^k, \quad k = 0, 1, 2, \ldots$$

Usage

```
bf.dist.negative_binomial(
  total_count,
  probs,
  logits = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`total_count`	The total number of events.
`probs`	See the Python documentation of `probs`.
`logits`	See the Python documentation of `logits`.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.

obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF NegativeBinomial distribution object (for model building).
- JAX array of samples drawn from the NegativeBinomial distribution (for direct sampling).
- The raw BF distribution object (for advanced use cases).

References

https://en.wikipedia.org/wiki/Negative_binomial_distribution

https://www.statology.org/negative-binomial-distribution/

See Also

https://num.pyro.ai/en/stable/distributions.html#negativebinomial

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.negative_binomial(
    total_count = 5.0,
    probs = jnp$array(c(0.2, 0.3, 0.5)),
    sample = TRUE
)
```

`bf.dist.negative_binomial2`
Negative Binomial2 distribution

Description

Samples from a Negative Binomial

The NB2 parameterisation of the negative-binomial distribution is a count distribution used for modelling over-dispersed count data (variance > mean). It is defined such that the variance grows **quadratically** in the mean:

$$mathrmVar(Y) = \mu + \alpha, \mu^2,$$

where (\mu = \mathbb{E}[Y]) and (\alpha>0) is the dispersion (heterogeneity) parameter. Because of this quadratic variance growth, it is called the NB2 family.

$$P(k) = \frac{\Gamma(k+\alpha)}{\Gamma(k+1)\Gamma(\alpha)} \left(\frac{\beta}{\alpha+\beta}\right)^k \left(1-\frac{\beta}{\alpha+\beta}\right)^k$$

One commonly used form of the NB2 parameterisation is obtained via a Poisson-Gamma mixture:

$$Y \mid \lambda \sim \text{Poisson}(\lambda), \quad \lambda \sim \text{Gamma!}\left(\frac{1}{\alpha}, , \frac{\mu, \alpha}{1}\right)$$

which marginalises to:

$$\Pr(Y=y) = \frac{\Gamma(y+\frac{1}{\alpha})}{\Gamma(\frac{1}{\alpha}); y!}; \left(\frac{1}{1+\alpha, \mu}\right)^{1/\alpha}; \left(\frac{\alpha, \mu}{1+\alpha, \mu}\right)^{y}, \quad y = 0, 1, 2, \ldots$$

This parameterisation matches the NB2 form with mean μ and dispersion α. (See Hilbe (2011) forderivation.)

Usage

```
bf.dist.negative_binomial2(
  mean,
  concentration,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

mean	The mean of the distribution. This is equivalent to the mu parameter.
concentration	The concentration parameter. This is equivalent to the $alpha$ parameter.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF NegativeBinomial distribution object: When sample = FALSE (for model building).
- numeric vector or array: A JAX array of samples drawn from the NegativeBinomial distribution (for direct sampling).
- BF NegativeBinomial distribution object: When create_obj = TRUE (for advanced use cases).

References

[Negative Binomial Regression]9https://www.cambridge.org/core/books/negative-binomial-regression/12D6281A46B9A98(

Negative Binomial Regression: Second Edition

See Also

https://num.pyro.ai/en/stable/distributions.html#negativebinomial

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.negative_binomial2(mean = 2.0, concentration = 3.0, sample = TRUE)
```

`bf.dist.negative_binomial_logits`

Negative Binomial Logits distribution

Description

Samples from a Negative Binomial Logits distribution.

The Negative Binomial Logits distribution is a generalization of the Negative Binomial distribution where the parameter 'r' (number of successes) is expressed as a function of a logit parameter. This allows for more flexible modeling of count data.

Usage

```
bf.dist.negative_binomial_logits(
  total_count,
  logits,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`total_count`	The parameter controlling the shape of the distribution. Represents the total number of trials.
`logits`	The log-odds parameter. Related to the probability of success.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.

obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- Negative Binomial Logits: A BF Negative Binomial Logits distribution object (for model building).
- numeric vector or array: A JAX array of samples drawn from the Negative Binomial Logits distribution (for direct sampling).
- Negative Binomial Logits: The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#Negative

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.negative_binomial_logits(total_count = 5.0, logits = 0.0, sample = TRUE)
```

bf.dist.negative_binomial_probs
Negative Binomial with probabilities. distribution

Description

The Negative Binomial distribution models the number of failures before the first success in a sequence of independent Bernoulli trials. It is characterized by two parameters: 'concentration' (r) and 'rate' (p). In this implementation, the 'concentration' parameter is derived from 'total_count' and the 'rate' parameter is derived from 'probs'.

Usage

```
bf.dist.negative_binomial_probs(
  total_count,
  probs,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

total_count	A numeric vector, matrix, or array representing the parameter.
probs	A numeric vector representing event probabilities. Must sum to 1.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.

seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF NegativeBinomialProbs distribution object (for model building).
- JAX array of samples drawn from the NegativeBinomialProbs distribution (for direct sampling).
- The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#negativebinomialprobs

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.negative_binomial_probs(
    total_count = 10.0,
    probs = jnp$array(c(0.2, 0.3, 0.5)),
    sample = TRUE
)
```

---

bf.dist.normal *Normal distribution*

Description

Samples from a Normal (Gaussian) distribution.

The Normal distribution is characterized by its mean (loc) and standard deviation (scale). It's a continuous probability distribution that arises frequently in statistics and probability theory.

$$f(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp! \left(-\frac{(x-\mu)^2}{2\sigma^2} \right)$$

Usage

```
bf.dist.normal(
  loc = 0,
  scale = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	The mean of the distribution.
scale	The standard deviation of the distribution.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When `sample = FALSE`: A BF Normal distribution object (for model building).
- When `sample = TRUE`: A JAX array of samples drawn from the Normal distribution (for direct sampling).
- When `create_obj = TRUE`: The raw BF distribution object (for advanced use cases).

References

Wikipedia: Normal distribution – PDF

Wolfram MathWorld: Normal Distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#normal

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.normal(loc = 0.0, scale = 1.0, sample = TRUE)
```

---

bf.dist.ordered_logistic

Ordered Logistic distribution

Description

The ordered logistic distribution is used for modeling **ordinal** outcome variables $Y \in 1, 2, \ldots, K$ (i.e., categories with a natural order) via a latent continuous predictor η and a set of increasing *cut-points* $c_1 < c_2 < \cdots < c_{K-1}$. When η crosses thresholds, the observed Y

PMF (Probability Mass Function):

Let $\eta \in \mathbb{R}$ be the linear predictor (for example $\eta = x^\top \beta$), let cutpoints $c = (c_1, \ldots, c_{K-1})$ satisfy $c_1 < c_2 < \cdots < c_{K-1}$. Define also $c_0 = -\infty$, $c_K = +\infty$. Then for (k \in {1,\dots,K}),

$$\Pr(Y = k \mid \eta, c) = \begin{cases} 1 - \text{logit}^{-1}(\eta - c_1), & k = 1, \\ \text{logit}^{-1}(\eta - c_{k-1}) - \text{logit}^{-1}(\eta - c_k), & 1 < k < K, \\ \text{logit}^{-1}(\eta - c_{K-1}), & k = K. \end{cases}$$

Here $\text{logit}^{-1}(z) = \frac{1}{1+e^{-z}}$. ([Stan][2]) In other words:

$$\Pr(Y \leq k \mid \eta, c) = \text{logit}^{-1}(\eta - c_k),$$

and so

$$\Pr(Y = k) = \Pr(Y \leq k) - \Pr(Y \leq k - 1).$$

This formulation appears in e.g. the CDF-link setup of the proportional odds model.

Usage

```
bf.dist.ordered_logistic(
  predictor,
  cutpoints,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  infer = py_none(),
  enum = FALSE,
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

predictor	Prediction in real domain; typically this is output of a linear model.
cutpoints	Positions in real domain to separate categories.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
infer	Optional named list passed to NumPyro's infer argument (e.g. list(enumerate = "parallel")).
enum	Logical. If TRUE, the discrete latent is summed out exactly (parallel enumeration) instead of sampled; same as infer = list(enumerate = "parallel"). Defaults to FALSE.
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.

seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF OrderedLogistic distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the OrderedLogistic distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

[1]: https://en.wikipedia.org/wiki/Ordered_logit

[2]: https://mc-stan.org/docs/2_21/functions-reference/ordered-logistic-distribution.html

[3]: https://labdisia.disia.unifi.it/grilli/files/Papers/Ordered_Logit.pdf"

See Also

https://num.pyro.ai/en/stable/distributions.html#orderedlogistic

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.ordered_logistic(
    predictor = jnp$array(c(0.2, 0.5, 0.8)),
    cutpoints = jnp$array(c(-1.0, 0.0, 1.0)),
    sample = TRUE
)
```

`bf.dist.pareto` *Pareto distribution*

Description

Samples from a Pareto distribution.

The **Pareto distribution**, named after economist Vilfredo Pareto, is a **power-law** probability distribution used to describe phenomena with "rich-get-richer" or "heavy-tail" properties — for example, income distribution, city sizes, or wealth concentration. It is characterized by:

- a **scale parameter** $x_m > 0$ (the minimum possible value), and
- a **shape parameter** $\alpha > 0$ (which controls the tail heaviness).

A random variable $X \sim \text{Pareto}(\alpha, x_m)$ takes values $x \geq x_m$.

$$f(x \mid \alpha, x_m) = \begin{cases} \dfrac{\alpha x_m^{\alpha}}{x^{\alpha+1}}, & x \geq x_m, \\ 0, & x < x_m. \end{cases}$$

Usage

```
bf.dist.pareto(
  scale,
  alpha,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`scale`	Scale parameter of the Pareto distribution. Must be positive.
`alpha`	Shape parameter of the Pareto distribution. Must be positive.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.

obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Pareto distribution object (for model
- building).
- When sample = TRUE: A JAX array of samples drawn from the Pareto
- distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced
- use cases).

References

Wikipedia: Pareto distribution – PDF

Arnold, B. C. (2015). *Pareto Distributions*, Second Edition. CRC Press.

See Also

https://num.pyro.ai/en/stable/distributions.html#pareto

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.pareto(scale = 2.0, alpha = 3.0, sample = TRUE)
```

`bf.dist.poisson` *Poisson distribution*

Description

The **Poisson distribution** models the probability of observing a given number of events k occurring in a fixed interval of time or space when these events happen independently and at a constant average rate $\lambda > 0$. It is widely used for modeling **count data**, such as the number of emails received per hour or mutations in a DNA strand per unit length.

Formally,

$$K \sim \text{Poisson}(\lambda)$$

where λ is both the **mean** and **variance** of the distribution.

The probability mass function (PMF) of the Poisson distribution is given by:

$$P(K = k \mid \lambda) = \frac{e^{-\lambda}\lambda^k}{k!}, \quad k = 0, 1, 2, \ldots$$

Usage

```
bf.dist.poisson(
  rate,
  is_sparse = FALSE,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`rate`	The rate parameter, representing the average number of events.
`is_sparse`	Indicates whether the `rate` parameter is sparse. If `TRUE`, a specialized sparse sampling implementation is used, which can be more efficient for models with many zero-rate components (e.g., zero-inflated models). Defaults to `FALSE`.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).

name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Poisson distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Poisson distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

Wikipedia: Poisson distribution – PMF

Johnson, N. L., Kotz, S., & Kemp, A. W. (1992). *Univariate Discrete Distributions* (2nd ed.). Wiley.

See Also

https://num.pyro.ai/en/stable/distributions.html#poisson

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.poisson(rate = 2.0, sample = TRUE)
```

`bf.dist.projected_normal` *Projected Normal distribution*

Description

The projected normal distribution arises by taking a multivariate normal vector $\mathbf{X} \sim \mathcal{N}_n(\mu, \Sigma)$ in $\mathbb{R}^n$ and projecting it to the unit sphere. This distribution is commonly used in directional statistics (data on circles or spheres) and supports asymmetric and even multimodal behaviours depending on parameters.

Usage

```
bf.dist.projected_normal(
  concentration,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`concentration`	The concentration parameter, representing the direction towards which the samples are concentrated. Must be a JAX array with at least one dimension.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
`obs`	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.
`mask`	Optional logical vector or array masking observations (`FALSE` = ignored). Defaults to `py_none()`.
`sample`	Logical. If `TRUE`, directly draw values from the distribution and return them; if `FALSE` (default), create a random variable inside a model.
`seed`	Integer seed for this draw when `sample = TRUE`. By default (`py_none()`) the draw comes from the session's random stream, set by `importBF(rand_seed = ...)`, so consecutive draws differ but the whole sequence is reproducible. No effect when `sample = FALSE`.

shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF ProjectedNormal distribution object (for
- model building).
- When sample = TRUE: A JAX array of samples drawn from the
- ProjectedNormal distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for
- advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#projectednormal

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.projected_normal(concentration = jnp$array(c(1.0, 3.0, 2.0)), sample = TRUE)
```

---

bf.dist.relaxed_bernoulli

Relaxed Bernoulli distribution

Description

The Relaxed Bernoulli is a continuous distribution on the interval $(0, 1)$ that smoothly approximates the discrete Bernoulli distribution (which has support $0, 1$). It was introduced to allow for *differentiable* sampling of approximate binary random variables, which is useful in variational inference and other gradient-based optimization settings.

The probability density function (PDF) is defined as:

- It has a *temperature* parameter $\tau > 0$ controlling the smoothness: as $\tau \to 0$, the distribution concentrates near $0, 1$ (thus approximating the discrete Bernoulli).

- For larger τ, the sample becomes more "soft" and less binary, e.g., converging toward 0.5 for very high τ.
- It is parameterised by a probability p (or equivalently logits $\ell = \log \frac{p}{1-p}$).

Let $X \in (0, 1)$ follow a Relaxed Bernoulli (Binary Concrete) distribution with location parameter $\alpha > 0$ (sometimes expressed via logits) and temperature parameter $\lambda > 0$. Then the PDF is:

$$p_{,\alpha,;\lambda}(x) = \frac{;\lambda, \alpha, x^{-\lambda-1}, (1-x)^{-\lambda-1};}{\left(\alpha, x^{-\lambda} + (1-x)^{-\lambda}\right)^2}, \quad \text{for } 0 < x < 1.$$

Usage

```
bf.dist.relaxed_bernoulli(
  temperature,
  probs = py_none(),
  logits = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

temperature	The temperature parameter. Must be greater than 0.
probs	The probability of success. Must be in the interval \[0, 1\]. Only one of probs or logits can be specified.
logits	The log-odds of success. Must be in the interval \[-inf, inf\]. Only one of probs or logits can be specified.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.

seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF RelaxedBernoulli distribution object (for model building) when sample = FALSE.
- A JAX array of samples drawn from the RelaxedBernoulli distribution (for direct sampling) when sample = TRUE.
- The raw BF distribution object (for advanced use cases) when create_obj = TRUE.

References

[1]: https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/RelaxedBernoulli "tfp.distributions.RelaxedBe
| TensorFlow Probability"

[2]: https://rstudio.github.io/tfprobability/reference/tfd_relaxed_bernoulli.html "RelaxedBernoulli distribution with temperature and logits ..."

See Also

https://num.pyro.ai/en/stable/distributions.html#relaxedbernoulli

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.relaxed_bernoulli(
    temperature = 1.0,
    probs = jnp$array(c(0.2, 0.3, 0.5)),
    sample = TRUE
)
```

`bf.dist.relaxed_bernoulli_logits`
Relaxed Bernoulli Logits distribution

Description

The Relaxed Bernoulli (logits) is a **continuous** relaxation of the standard Bernoulli distribution, parameterised by logits (or probabilities) and a **temperature** parameter. Rather than producing strictly 0 or 1, it produces values in the continuous interval (0, 1). As the temperature -> 0 the distribution approximates a Bernoulli; as temperature -> ∞ the distribution approximates a uniform distribution.

It is used in variational inference and deep-learning contexts to allow gradient-based optimisation through otherwise discrete Bernoulli draws (see the "Concrete" or "Gumbel-Softmax" literature: The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables and Categorical Reparameterization with Gumbel-Softmax).

Because this is a continuous relaxation, the "density" is defined over the unit interval (0,1) rather than a pmf at {0,1}. Using logistic or Gumbel-softmax style construction one can express it as:

If logits = (\ell), and temperature = (\tau > 0), then one generates

$$u \sim \mathrm{Uniform}(0,1), \quad g = -\log(-\log(u)), \quad x = \sigma\big((\ell + g)/\tau\big)$$

where $\sigma(\cdot)$ is the logistic (sigmoid) function. Then $x \in (0,1)$ has the RelaxedBernoulli distribution.

The exact density formula can be derived via change of variables from logistic/Gumbel, but is somewhat involved (including Jacobian of sigmoid transform).

Usage

```
bf.dist.relaxed_bernoulli_logits(
  temperature,
  logits,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

temperature	The temperature parameter, must be positive.
logits	The logits parameter.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- RelaxedBernoulliLogits: A BF RelaxedBernoulliLogits distribution object (for model building).
- numeric vector or array: A JAX array of samples drawn from the RelaxedBernoulliLogits distribution (for direct sampling).
- RelaxedBernoulliLogits: The raw BF distribution object (for advanced use cases).

References

https://arxiv.org/pdf/1611.00712

See Also

https://num.pyro.ai/en/stable/distributions.html#relaxed-bernoulli-logits

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.relaxed_bernoulli_logits(temperature = 1.0, logits = 0.0, sample = TRUE)
```

`bf.dist.right_truncated_distribution`

Right Truncated distribution

Description

A right-truncated distribution is a statistical distribution that arises when the possible values of a random variable are restricted to be below a certain specified value `high`. In essence, the right tail of the original distribution is "cut off" at a particular point, and the remaining probability is redistributed among the allowable values. This type of distribution is common in various fields where there are inherent upper limits or observational constraints.

The probability density function (PDF) for a continuous right-truncated distribution or the probability mass function (PMF) for a discrete one is derived from the original distribution by normalizing it over the restricted range.

For a continuous random variable X:

If the original probability density function is `f(x)` and the cumulative distribution function is `F(x)`, and the distribution is right-truncated at a value `b` (meaning `x <= b`), the new PDF, `f_T(x)`, is:

$$f_{\text{trunc}}(x \mid X \leq b) = \begin{cases} ,f(x), \\ \overline{,F(b),}, & x \leq b, \\ 0, & x > b. \end{cases}$$

For a discrete random variable X:

If the original probability mass function is `P(X = k)` and the cumulative distribution function is `F(k)`, and the distribution is right-truncated at a value `b` (meaning `k <= b`), the new PMF, `P_T(X = k)`, is:

$$P_T(X = k) = \begin{cases} \frac{P(X=k)}{F(b)} & \text{if } k \leq b \\ 0 & \text{if } k > b \end{cases}$$

Usage

```
bf.dist.right_truncated_distribution(
  base_dist,
  high = 0,
  validate_args = py_none(),
  name = "x",
```

```
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

base_dist	The base distribution to truncate. Must be a univariate distribution with real support.
high	The upper truncation point. The support of the new distribution is $(-\infty, \text{high}]$. Defaults to 0.0.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF RightTruncatedDistribution distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the RightTruncatedDistribution distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

https://en.wikipedia.org/wiki/Truncated_normal_distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#righttruncateddistribution

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.right_truncated_distribution(
    base_dist = bf.dist.normal(0, 1, create_obj = TRUE),
    high = 0,
    sample = TRUE
)
```

---

bf.dist.soft_laplace *SoftLaplace distribution*

Description

Smooth distribution with Laplace-like tail behavior.

This distribution corresponds to the log-convex density:

$$z = (value - loc)/scalelog_prob = log(2/pi) - log(scale) - logaddexp(z, -z)$$

Like the Laplace density, this density has the heaviest possible tails (asymptotically) while still being log-convex. Unlike the Laplace distribution, this distribution is infinitely differentiable everywhere, and is thus suitable for HMC and Laplace approximation.

Usage

```
bf.dist.soft_laplace(
  loc,
  scale,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
```

```
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	Location parameter.
scale	Scale parameter.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF SoftLaplace distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the SoftLaplace distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#softlaplace

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.soft_laplace(loc = 0.0, scale = 1.0, sample = TRUE)
```

`bf.dist.student_t` *Student's t distribution*

Description

The Student's t-distribution is a probability distribution that arises in hypothesis testing involving the mean of a normally distributed population when the population standard deviation is unknown. It is similar to the normal distribution, but has heavier tails, making it more robust to outliers. For large ν, it converges to the Normal distribution.

$$X \sim t_\nu(\mu, \sigma)$$

where:

- μ is the **location (mean)** parameter
- $\sigma > 0$ is the **scale** parameter
- $\nu > 0$ is the **degrees of freedom** controlling the tail heaviness

The probability density function (pdf) of the Student's t distribution is:

$$f(x \mid \nu, \mu, \sigma) = \frac{\Gamma!\left(\frac{\nu+1}{2}\right)}{\Gamma!\left(\frac{\nu}{2}\right)\sqrt{\pi\nu}\sigma}\left[1 + \frac{1}{\nu}\left(\frac{x-\mu}{\sigma}\right)^2\right]^{-\frac{\nu+1}{2}}$$

- The heavier tails (for small ν) allow for larger outliers.
- For $\nu \to \infty$, this approaches $\mathcal{N}(\mu, \sigma^2)$.

Usage

```
bf.dist.student_t(
  df,
  loc = 0,
  scale = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
```

```
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

df	Degrees of freedom, must be positive.
loc	Location parameter, defaults to 0.0.
scale	Scale parameter, defaults to 1.0.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF StudentT distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the StudentT distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

Wikipedia – Student's t distribution: PDF

See Also

https://num.pyro.ai/en/stable/distributions.html#studentt

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.student_t(df = 2, loc = 0.0, scale = 1.0, sample = TRUE)
```

`bf.dist.truncated_cauchy`

Truncated Cauchy distribution

Description

The Cauchy distribution, also known as the Lorentz distribution, is a continuous probability distribution that appears frequently in various areas of mathematics and physics. It is characterized by its heavy tails, which extend to infinity. The truncated version limits the support of the Cauchy distribution to a specified interval.

Let the base Cauchy have parameters μ (location) and $\gamma > 0$ (scale). Let the truncation bounds be a (`low`) and b (`high`) (with $a < b$). Then the PDF of the truncated Cauchy at $x \in [a, b]$ is:

$$f_T(x) = \frac{1}{\pi, \gamma}; \frac{1}{1 + \left(\frac{x-\mu}{\gamma}\right)^2}; \Big/ ; \left[F_{\text{Cauchy}}!\left(\frac{b-\mu}{\gamma}\right) - F_{\text{Cauchy}}!\left(\frac{a-\mu}{\gamma}\right)\right],$$

and (f_{T}(x)=0) for (x < a) or (x > b).

Here F_{Cauchy} is the standard Cauchy CDF $withparameters$\mu,\gamma)$.

Usage

```
bf.dist.truncated_cauchy(
  loc = 0,
  scale = 1,
  low = py_none(),
  high = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
```

```
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	Location parameter of the Cauchy distribution.
scale	Scale parameter of the Cauchy distribution.
low	See the Python documentation of low.
high	See the Python documentation of high.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF TruncatedCauchy distribution object (for model building) when sample = FALSE.
- JAX array of samples drawn from the TruncatedCauchy distribution (for direct sampling) when sample = TRUE.
- The raw BF distribution object (for advanced use cases) when create_obj = TRUE.

References

[1]: https://en.wikipedia.org/wiki/Cauchy_distribution "Cauchy distribution"

[2]: https://www.tandfonline.com/doi/full/10.1080/00207390600595223 "Full article: A truncated Cauchy distribution"

[3]: https://rstudio.github.io/tfprobability/reference/tfd_truncated_cauchy.html "The Truncated Cauchy distribution. — tfd_truncated_cauchy"

See Also

https://num.pyro.ai/en/stable/distributions.html#truncatedcauchy

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.truncated_cauchy(loc = 0.0, scale = 1.0, sample = TRUE)
```

---

bf.dist.truncated_distribution

Truncated distribution

Description

Samples from a Truncated Distribution.

A **truncated distribution** arises when you take a random variable X that originally has some distribution (with PDF $f_X(x)$ and CDF $F_X(x)$) and you restrict attention only to those values of X that are *above* a given truncation point a. In other words you only observe X when $X > a$. All the "mass" below (or equal to) a is **excluded** (not just unobserved, but removed from the sample/analysis). This differs from *censoring*, where values below a threshold might be known (for example "< a"), but here they are entirely excluded from the domain. Left truncation is common in many applied fields — for instance:

- In survival analysis: subjects whose event time happens before the study start are not included.
- In insurance or losses: only losses above a deductible (threshold) are recorded, so the loss distribution is left-truncated at that deductible.
- In industrial/life-data: items used before data collection start, so only those with lifetime > some lower bound are observed.

Let (X) be a random variable with PDF. Choose truncation points $a < X \leq b$. Then the truncated distribution (Y = X \mid (a < X \le b)) has:

$$f_Y(y) = \frac{f_X(y)}{F_X(b) - F_X(a)}, \quad \text{for } a < y \leq b,$$

and (0) outside that interval.

- The lower bound (a) can be (-\infty) (so only an upper truncation), and the upper bound (b) can be (+\infty) (so only a lower truncation). ([Statistics How To][2])
- If both bounds are finite (i.e., doubly truncated), that is a valid and often-used scenario.

Usage

```
bf.dist.truncated_distribution(
  base_dist,
  low = py_none(),
  high = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

base_dist	The base distribution to be truncated. This should be a univariate distribution. Currently, only the following distributions are supported: Cauchy, Laplace, Logistic, Normal, and StudentT.
low	See the Python documentation of low.
high	See the Python documentation of high.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).

event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF TruncatedDistribution distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the TruncatedDistribution distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

[1]: https://en.wikipedia.org/wiki/Truncated_distribution "Truncated distribution"

[2]: https://pages.stern.nyu.edu/~wgreene/Text/Edition8/PDF/M19_GREE1366_08_SE_C19.pdf "19"

[3]: https://search.r-project.org/CRAN/refmans/LaplacesDemon/html/dist.Truncated.html "R: Truncated Distributions"

[4]: https://dspace.cuni.cz/bitstream/handle/20.500.11956/127921/130308341.pdf?isAllowed=y&sequence=1&utm_source. "Truncated data thesis"

See Also

https://num.pyro.ai/en/stable/distributions.html#truncateddistribution

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.truncated_distribution(
    base_dist = bf.dist.normal(0, 1, create_obj = TRUE),
    high = 1,
    low = 0,
    sample = TRUE
)
```

`bf.dist.truncated_normal`

Truncated Normal distribution

Description

A truncated normal distribution is derived from a normal (Gaussian) random variable by restricting (truncating) its domain to an interval $[a, b]$ (which could be one-sided, e.g., (a) only or (b) only). It is defined by its location (`loc`), scale (`scale`), lower bound a (`low`), and upper bound b (`high`). In effect: if $X \sim \mathcal{N}(\mu, \sigma^2)$, then the truncated version $Y = X|(a \leq X \leq b)$ has the same "shape" but only supports values in $[a, b]$. This is used when you know that values outside a range are impossible or not observed (e.g., measurement limits, natural bounds).

Let $X \sim \mathcal{N}(\mu, \sigma^2)$. Define truncation bounds $a < b$ (could be $a = -\infty$ or $b = +\infty$ for one-sided truncation). Then for $y \in [a, b]$,

$$f_Y(y) = \frac{1}{\sigma}; \frac{\varphi!\left(\frac{y-\mu}{\sigma}\right)}{\Phi!\left(\frac{b-\mu}{\sigma}\right) - \Phi!\left(\frac{a-\mu}{\sigma}\right)},$$

where:

- $\varphi(z)$ is the standard normal PDF, $\varphi(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$.
- $\Phi(z)$ is the standard normal CDF.
- The denominator $\Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)$ normalises the density so the total probability over ([a,b]) is 1.

For values (y < a) or (y > b), (f_Y(y) = 0).

Usage

```
bf.dist.truncated_normal(
  loc = 0,
  scale = 1,
  low = py_none(),
  high = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

loc	The location parameter of the normal distribution.
scale	The scale parameter of the normal distribution.
low	See the Python documentation of low.
high	See the Python documentation of high.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF TruncatedNormal distribution object (for model building).
- JAX array of samples drawn from the TruncatedNormal distribution (for
- direct sampling).
- The raw BF distribution object (for advanced use cases).

References

[1]: https://en.wikipedia.org/wiki/Truncated_normal_distribution "Truncated normal distribution"

[2]: https://people.math.sc.edu/burkardt/m_src/truncated_normal/truncated_normal.html "TRUNCATED_NORMAL - The Truncated Normal Distribution"

[3]: https://www.statisticshowto.com/truncated-normal-distribution/ "Truncated Distribution / Truncated Normal Distribution - Statistics ..."

[4]: https://real-statistics.com/normal-distribution/truncated-normal-distribution/ "Truncated Normal Distribution | Real Statistics Using Excel"

[5]: https://stats.stackexchange.com/questions/525894/understanding-the-pdf-of-a-truncated-normal-distribution "Understanding the pdf of a truncated normal distribution"

See Also

https://num.pyro.ai/en/stable/distributions.html#truncatednormal_lowercase

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.truncated_normal(loc = 0.0, scale = 1.0, sample = TRUE)
```

---

bf.dist.truncated_polya_gamma

Truncated PolyaGamma distribution

Description

Samples from a Truncated PolyaGamma distribution.

This distribution is a truncated version of the PolyaGamma distribution, defined over the interval [0, truncation_point]. It is often used in Bayesian non-parametric models.

Usage

```
bf.dist.truncated_polya_gamma(
  batch_shape = c(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

batch_shape	The shape of the batch dimension.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Truncated PolyaGamma distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Truncated PolyaGamma distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#truncatedpolygammadistribution

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.truncated_polya_gamma(batch_shape = c(), sample = TRUE)
```

`bf.dist.two_sided_truncated_distribution`
Two Sided Truncated distribution

Description

A "two-sided truncated distribution" is a general concept: you take a base continuous distribution and **restrict it** to an interval ([`low`, `high`]), discarding all mass outside, then **renormalize** so the inner portion integrates to 1. I'll spell out the general formulas, caveats, sampling strategies, and special cases (e.g. truncated normal) to illustrate.

$$f(x) = \begin{cases} \dfrac{p(x)}{P(\text{low} \le X \le \text{high})}, & \text{if low} \le x \le \text{high}, \\ 0, & \text{otherwise.} \end{cases}$$

where $p(x)$ is the probability density function of the base distribution.

Usage

```
bf.dist.two_sided_truncated_distribution(
  base_dist,
  low = 0,
  high = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`base_dist`	The base distribution to truncate.
`low`	The lower bound for truncation.
`high`	The upper bound for truncation.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.

obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF TwoSidedTruncatedDistribution distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the TwoSidedTruncatedDistribution distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

See Also

https://num.pyro.ai/en/stable/distributions.html#twosidedtruncateddistribution

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.two_sided_truncated_distribution(
    base_dist = bf.dist.normal(0.0, 1.0, create_obj = TRUE),
    low = -1.0,
    high = 1.0,
    sample = TRUE
)
```

`bf.dist.uniform` *Uniform distribution*

Description

The Uniform distribution is the simplest continuous distribution: every value in the interval ([a, b]) is **equally likely**. It is widely used for modeling complete randomness within a fixed range, random sampling, and as a building block for other distributions.

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \le x \le b, \\ 0, & \text{otherwise.} \end{cases}$$

Usage

```
bf.dist.uniform(
  low = 0,
  high = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`low`	The lower bound of the uniform interval.
`high`	The upper bound of the uniform interval.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
`obs`	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.
`mask`	Optional logical vector or array masking observations (`FALSE` = ignored). Defaults to `py_none()`.
`sample`	Logical. If `TRUE`, directly draw values from the distribution and return them; if `FALSE` (default), create a random variable inside a model.

seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF Uniform distribution object (for model building) when sample = FALSE.
- JAX array of samples drawn from the Uniform distribution (for direct
- sampling) when sample = TRUE.
- The raw BF distribution object (for advanced use cases) when
- create_obj = TRUE.

References

Wikipedia – Uniform distribution (continuous)

See Also

https://num.pyro.ai/en/stable/distributions.html#uniform

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.uniform(low = 0.0, high = 1.0, sample = TRUE)
```

`bf.dist.weibull` *Weibull distribution*

Description

Samples from a Weibull distribution.

The Weibull distribution is widely used for modeling **lifetime or reliability data**. Its shape parameter (k) controls the hazard function:

- (k < 1): decreasing hazard (infant mortality)
- (k = 1): constant hazard -> reduces to **Exponential distribution**
- (k > 1): increasing hazard (aging/failure over time)

$$f(x \mid \lambda, k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, & x \geq 0, \\ 0, & x < 0 \end{cases}$$

Usage

```
bf.dist.weibull(
  scale,
  concentration,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

`scale`	The scale parameter of the Weibull distribution. Must be positive.
`concentration`	The concentration parameter of the Weibull distribution. Must be positive.
`validate_args`	Logical. Whether to validate the parameter values. Defaults to `py_none()` (the NumPyro default).
`name`	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to `obs` (`obs = y` gives `"y"`); use `bf_model()` to name parameters after the variables they are assigned to.
`obs`	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to `py_none()`.

mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF Weibull distribution object (for model building) when sample = FALSE.
- JAX array of samples drawn from the Weibull distribution (for direct sampling) when sample = TRUE.
- The raw BF distribution object (for advanced use cases) when create_obj = TRUE.

References

Wikipedia – Weibull distribution

Weibull, W. (1951). *A Statistical Distribution Function of Wide Applicability.* Journal of Applied Mechanics.

See Also

https://num.pyro.ai/en/stable/distributions.html#weibull

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.weibull(scale = 1.0, concentration = 2.0, sample = TRUE)
```

`bf.dist.wishart` *Wishart distribution*

Description

The Wishart distribution is a multivariate distribution used to model positive definite matrices, often representing covariance matrices. It's commonly used in Bayesian statistics and machine learning, particularly in models involving covariance estimation.

$$\mathbf{X} \sim \text{Wishart}(\text{scale_matrix} = \mathbf{V}, \text{concentration} = n)$$

where:

- (p) = dimension of the square matrices
- (\mathbf{V} \in \mathbb{R}^{p \times p}) = positive definite **scale matrix** (`scale_matrix`)
- (n \ge p) = **degrees of freedom** (`concentration`)

The Wishart distribution is the **multivariate generalization of the chi-squared distribution**. It is commonly used as the distribution of the **sample covariance matrix** for multivariate normal samples.

$$f(\mathbf{X} \mid \mathbf{V}, n) = \frac{|\mathbf{X}|^{(n-p-1)/2} \exp\left(-\frac{1}{2}\text{tr}(\mathbf{V}^{-1}\mathbf{X})\right)}{2^{np/2}|\mathbf{V}|^{n/2}\Gamma_p(n/2)}, \quad \mathbf{X} \succ 0$$

where:

- $|\mathbf{X}|$ = determinant of $\mathbf{X}$
- $\text{tr}(\cdot)$ = trace
- $\Gamma_p(\cdot)$ = multivariate Gamma function:

Usage

```
bf.dist.wishart(
  concentration,
  scale_matrix = py_none(),
  rate_matrix = py_none(),
  scale_tril = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

concentration	Positive concentration parameter analogous to the concentration of a Gamma distribution. The concentration must be larger than the dimensionality of the scale matrix.
scale_matrix	Scale matrix analogous to the inverse rate of a Gamma distribution.
rate_matrix	Rate matrix anaologous to the rate of a Gamma distribution.
scale_tril	Cholesky decomposition of the :code:scale_matrix.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF Wishart distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the Wishart distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

Wikipedia – Wishart distribution

See Also

https://num.pyro.ai/en/stable/distributions.html#wishart

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.wishart(concentration = 5.0, scale_matrix = jnp$eye(2L), sample = TRUE)
```

`bf.dist.wishart_cholesky`

Wishart Cholesky distribution

Description

- The **Wishart** distribution is a distribution over positive definite matrices, often used as a prior for covariance or precision matrices in multivariate normal models.
- The **Cholesky parameterization** of the Wishart (called "wishart_cholesky" in Stan, for example) reparameterizes the Wishart over its **lower (or upper) triangular Cholesky factor**. This is useful for numerical stability and unconstrained parameterization in Bayesian sampling frameworks.
- In this parameterization, one works with a lower-triangular matrix L_W such that

$$\Sigma = L_W L_W^\top$$

and imposes a density over L_W corresponding to the induced Wishart density on (\Sigma).

The probability density function (PDF) is given by:

If the dimension is K, degrees of freedom $\nu > K - 1$, and L_S is the lower-triangular Cholesky factor of the scale matrix S, then the log density for L_W under the **wishart_cholesky** distribution is:

$$\log p(L_W \mid \nu, L_S) = \log\big[\text{Wishart}(L_W L_W^\top \mid \nu, L_S L_S^\top)\big] + \log\big|J_{f^{-1}}\big|$$

where $J_{f^{-1}}$ is the Jacobian of the transformation from L_W to $\Sigma = L_W L_W^\top$.

The Jacobian term (absolute log determinant) is:

$$\log|J_{f^{-1}}| = K\log 2; \sum_{k=1}^{K}(K-k+1), \log L_{W_{k,k}}.$$

Thus the density includes both the usual Wishart density on Σ plus this extra Jacobian (which penalizes or weights the diagonal entries of L_W).

Usage

```
bf.dist.wishart_cholesky(
  concentration,
  scale_matrix = py_none(),
  rate_matrix = py_none(),
  scale_tril = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

concentration	See the Python documentation of concentration.
scale_matrix	See the Python documentation of scale_matrix.
rate_matrix	See the Python documentation of rate_matrix.
scale_tril	See the Python documentation of scale_tril.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.

`create_obj`	Logical. If `TRUE`, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to `FALSE`.
`to_jax`	Logical. Return a JAX array (`TRUE`, default) or convert to R.

Value

A random variable (model building) or draws (`sample = TRUE`).

References

[1]: https://en.wikipedia.org/wiki/Wishart_distribution "Wishart distribution"

[2]: https://mc-stan.org/docs/2_32/functions-reference/wishart-cholesky-distribution.html"28.2 Wishart distribution, Cholesky Parameterization - Stan"

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.wishart_cholesky(concentration = 5.0, scale_matrix = jnp$eye(2L), sample = TRUE)
```

---

bf.dist.zero_inflated_distribution

Generic Zero Inflated distribution

Description

A Zero-Inflated distribution combines a base distribution with a Bernoulli distribution to model data with an excess of zero values. It assumes that each observation is either drawn from the base distribution or is a zero with probability determined by the Bernoulli distribution (the "gate"). A **zero-inflated distribution** arises when you take a random variable X that originally has some distribution (with PMF $f_X(x)$) *and* you add extra mass at zero. In other words you only observe X when $X \geq 0$ (if count) and you assume there are more zeros than what the original distribution predicts — so you mix in a point-mass at zero. A zero-inflated model assumes two processes: a "structural zero" process with probability π, and another process (the base count/distribution) with probability $1 - \pi$, which itself may generate zeros *or* non-zeros.

- When / Why use it:

- **Data collection constraints**: You can only observe values in a process when some condition is met; others are automatically zero (for example, when the event cannot happen for some units).
- **Mixtures of processes**: Some observations are "structural zeros" (no risk) and others follow a regular count process (with risk).

- **Overdispersion & excess zeros**: If you try a standard count distribution (Poisson, Negative Binomial) and you observe many more zeros than predicted (given the non-zero counts), a zero-inflated alternative may fit better.
- The general PMF form is (for discrete count models):

$$P(X = 0) = \pi + (1 - \pi), P_{\text{base}}(0), \quad P(X = k) = (1 - \pi), P_{\text{base}}(k), ; k > 0.$$

Usage

```
bf.dist.zero_inflated_distribution(
  base_dist,
  gate = py_none(),
  gate_logits = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

base_dist	The base distribution to be zero-inflated (e.g., Poisson, NegativeBinomial).
gate	Probability of extra zeros (between 0 and 1).
gate_logits	Log-odds of extra zeros.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.

shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When sample = FALSE: A BF ZeroInflatedDistribution distribution object (for model building).
- When sample = TRUE: A JAX array of samples drawn from the ZeroInflatedDistribution distribution (for direct sampling).
- When create_obj = TRUE: The raw BF distribution object (for advanced use cases).

References

[1]: https://en.wikipedia.org/wiki/Zero-inflated_model "Zero-inflated model"

[2]: https://larmarange.github.io/guide-R/analyses_avancees/modeles-zero-inflated.html "48 Modèles de comptage zero-inflated et hurdle – guide-R"

See Also

https://num.pyro.ai/en/stable/distributions.html#zeroinflateddistribution

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.zero_inflated_distribution(
    base_dist = bf.dist.poisson(rate = 5, create_obj = TRUE),
    gate = 0.3,
    sample = TRUE
)
```

`bf.dist.zero_inflated_negative_binomial`
Zero-Inflated Negative Binomial distribution

Description

A Zero-Inflated Negative Binomial distribution is used for count data that exhibit **both** (a) over-dispersion relative to a Poisson (i.e., variance > mean) *and* (b) an excess of zero counts beyond what a standard Negative Binomial would predict. It assumes two latent processes:

1. With probability π (sometimes denoted ψ or "zero-inflation probability") you are in a "structural zero" state -> you observe a zero.
2. With probability $1 - \pi$, you come from a regular Negative Binomial distribution (with parameters e.g. mean μ and dispersion parameter α or size/r parameter) and then you might observe zero or a positive count.

Thus the model is a mixture of a point-mass at zero + a Negative Binomial for counts.

This distribution combines a Negative Binomial distribution with a binary gate variable. Observations are either drawn from the Negative Binomial distribution with probability (1 - gate) or are treated as zero with probability 'gate'. This models data with excess zeros compared to what a standard Negative Binomial distribution would predict.

Let X denote the count random variable, support $0, 1, 2, \ldots$. Denote:

- π = probability of being in the "always zero" (structural zero) process, with $0 \leq \pi \leq 1$.
- The count process is NB with parameters: mean $\mu > 0$ and dispersion/shape parameter $\alpha > 0$ (or equivalently size parameter (r)). Then:

$$P(X = 0) = \pi; +; (1 - \pi); P_{\text{NB}}(0; \mu, \alpha),$$

$$P(X = k) = (1 - \pi); P_{\text{NB}}(k; \mu, \alpha), \quad k = 1, 2, 3, \ldots$$

Here $P_{\text{NB}}(k; \mu, \alpha)$ is the PMF of the negative binomial distribution for count k.

Usage

```
bf.dist.zero_inflated_negative_binomial(
  mean,
  concentration,
  gate = py_none(),
  gate_logits = py_none(),
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
```

```
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)

bf.dist.zero_inflated_negative_binomial2(...)
```

Arguments

mean	The mean of the Negative Binomial 2 distribution.
concentration	The concentration parameter of the Negative Binomial 2 distribution.
gate	See the Python documentation of gate.
gate_logits	See the Python documentation of gate_logits.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.
...	For bf.dist.zero_inflated_negative_binomial2(), the arguments of bf.dist.zero_inflated_ne (same function, Python name).

Value

- When sample = FALSE: A BF ZeroInflatedNegativeBinomial2 distribution object (for model building).

- When `sample = TRUE`: A JAX array of samples drawn from the ZeroInflatedNegativeBinomial2 distribution (for direct sampling).
- When `create_obj = TRUE`: The raw BF distribution object (for advanced use cases).

References

[1]: https://www.ewadirect.com/proceedings/tns/article/view/27624 "Social Networks Count Data: Negative Binomial Distributions and Extensions Versus Poisson and Bernoulli Models"

[2]: https://pmc.ncbi.nlm.nih.gov/articles/PMC7880198/ "Bayesian Zero-Inflated Negative Binomial Regression Based on Pólya-Gamma Mixtures - PMC"

[3]: https://search.r-project.org/CRAN/refmans/emdbook/html/dzinbinom.html "R: Zero-inflated negative binomial distribution"

See Also

https://num.pyro.ai/en/stable/distributions.html#zeroinflatednegativebinomial2

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.zero_inflated_negative_binomial(
    mean = 2.0,
    concentration = 3.0,
    gate = 0.3,
    sample = TRUE
)
```

---

bf.dist.zero_inflated_poisson

A Zero Inflated Poisson distribution

Description

The Zero-Inflated Poisson distribution is a discrete count-distribution designed for data with *more zeros* than would be expected under a standard Poisson. Essentially, it assumes two underlying processes:

- With probability π you are in a "structural zero" state (i.e., you automatically get a zero count).
- With probability $1 - \pi$ you draw from a standard Poisson distribution with parameter λ.

This results in a mixture distribution that places more mass at zero than a Poisson alone would. It's widely used in, for instance, ecology (species counts with many zeros), insurance/claims problems, and any count-data setting with excess zeros.

The probability density function (PDF) is given by:

Let X be a random variable following a Zero-Inflated Poisson with parameters

- $\pi \in [0, 1]$: zero-inflation probability (chance of being structural zero)
- $\lambda > 0$: mean of the Poisson component

Then:

$$P(X = k) = \begin{cases} \pi; +; (1 - \pi), e^{-\lambda}, & \text{if } k = 0, \\ (1 - \pi), \dfrac{\lambda^k e^{-\lambda}}{k!}, & \text{if } k = 1, 2, 3, \ldots \end{cases}$$

(Here the Poisson component contributes also a mass at zero of $(1 - \pi)e^{-\lambda}$, plus the extra π mass.)

Usage

```
bf.dist.zero_inflated_poisson(
  gate,
  rate = 1,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

gate	The probability of observing a zero.
rate	The rate parameter of the underlying Poisson distribution.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.

shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- BF ZeroInflatedPoisson distribution object (when sample = FALSE).
- JAX array of samples drawn from the ZeroInflatedPoisson distribution (when sample = TRUE).
- The raw BF distribution object (when create_obj = TRUE).

References

https://en.wikipedia.org/wiki/Zero-inflated_model "Zero-inflated model"

https://www.statisticshowto.com/zero-inflated-poisson-distribution "Zero-Inflated Poisson distribution - Statistics How To"

See Also

https://num.pyro.ai/en/stable/distributions.html#zeroinflatedpoisson

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.zero_inflated_poisson(gate = 0.3, rate = 2.0, sample = TRUE)
```

---

bf.dist.zero_sum_normal

Zero Sum Normal distribution

Description

A **zero-sum normal** is a variant of a multivariate normal in which one (or more) linear constraint(s) force certain components to **sum to zero**. In practice, it's used to model vectors of random effects (e.g. in hierarchical models) where the effects are constrained to sum to zero (to avoid overparameterization or enforce identifiability).

$$ZSN(\sigma) = N(0, \sigma^2(I - \tfrac{1}{n}J)) \text{where } \; J_{ij} = 1 \; \text{and} n = \text{number of zero-sum axes}$$

where $J$is the all-ones matrix and n is the number of constrained elements. So the covariance is $\sigma^2(I - \frac{1}{n}J)$, which ensures that the sum of the components is 0 (since $J\mathbf{x} = \mathbf{1}\sum_i x_i$).

Usage

```
bf.dist.zero_sum_normal(
  scale,
  event_shape,
  validate_args = py_none(),
  name = "x",
  obs = py_none(),
  mask = py_none(),
  sample = FALSE,
  seed = py_none(),
  shape = c(),
  event = 0,
  create_obj = FALSE,
  to_jax = TRUE
)
```

Arguments

scale	Standard deviation of the underlying normal distribution before the zerosum constraint is enforced.
event_shape	Shape of the event dimension of the distribution.
validate_args	Logical. Whether to validate the parameter values. Defaults to py_none() (the NumPyro default).
name	Character. Name of the random variable in the model. When not given, an observed site is named after the variable passed to obs (obs = y gives "y"); use bf_model() to name parameters after the variables they are assigned to.
obs	Numeric vector or array of observed values. If given, the random variable is conditioned on them (likelihood); otherwise it is latent. Defaults to py_none().
mask	Optional logical vector or array masking observations (FALSE = ignored). Defaults to py_none().
sample	Logical. If TRUE, directly draw values from the distribution and return them; if FALSE (default), create a random variable inside a model.
seed	Integer seed for this draw when sample = TRUE. By default (py_none()) the draw comes from the session's random stream, set by importBF(rand_seed = ...), so consecutive draws differ but the whole sequence is reproducible. No effect when sample = FALSE.
shape	Integer vector. With sample = FALSE, the batch shape of the distribution in the model; with sample = TRUE, the shape of the draw. Defaults to c() (no extra dimensions).
event	Integer. Number of batch dimensions reinterpreted as event dimensions (model building only). Defaults to 0.
create_obj	Logical. If TRUE, return the distribution object itself instead of a random variable (e.g. to build mixtures). Defaults to FALSE.
to_jax	Logical. Return a JAX array (TRUE, default) or convert to R.

Value

- When `sample = FALSE`: A BF ZeroSumNormal distribution object (for model building).
- When `sample = TRUE`: A JAX array of samples drawn from the ZeroSumNormal distribution (for direct sampling).
- When `create_obj = TRUE`: The raw BF distribution object (for advanced use cases).

References

https://www.pymc.io/projects/docs/en/latest/api/distributions/generated/pymc.ZeroSumNormal.html

See Also

https://num.pyro.ai/en/stable/distributions.html#zerosumnormal

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
bf.dist.zero_sum_normal(scale = 1.0, event_shape = c(2L), sample = TRUE)
```

---

bf.doc *title Open the BF Documentation Website*

Description

This function opens the BF documentation webpage in the user's default web browser.

Usage

```
bf.doc()
```

Value

Invisibly returns `NULL`. The function is called for its side effect of opening a URL.

bf_available *Is the BayesForge Python environment available?*

Description

TRUE when the 'BFR' virtual environment exists and is up to date (see setup_env()), or when a Python chosen by the user (RETICULATE_PYTHON, or one already initialised) has the BayesForge module. It never installs anything and does not start Python when none is configured, so it can guard examples and tests.

Usage

```
bf_available()
```

Value

A single logical value.

Examples

```
bf_available()
```

bf_draws *Posterior draws as an R data frame*

Description

Extracts the posterior draws of a fitted BayesForge model into a plain R data.frame, so they can be summarised, plotted or combined with any other R tool (dplyr, ggplot2, bayesplot, loo, ...).

Usage

```
bf_draws(m, format = c("data.frame", "draws_df", "long"), labels = NULL)

## S3 method for class 'BayesForge.Main.main.bf'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

m	A fitted BayesForge object (from importBF() followed by m$fit()).
format	"data.frame" (default, one row per draw), "draws_df", which returns a posterior::draws_df (requires the **posterior** package), or "long": one row per draw and parameter element, with columns .chain, .iteration, .draw, .variable (parameter name), .index (element, e.g. "2" or "1,3"; NA for a scalar) and .value – the tidy layout used by ggplot2 / dplyr / tidybayes.

labels — For `format = "long"`: a named list replacing the index of a vector parameter by labels, e.g. `list(alpha = levels(df$group))` when `alpha` is indexed by the 0-based codes of factor `group` (from `m$index()`).

x — A fitted BayesForge object.

row.names, optional, ... — Ignored; for compatibility with `as.data.frame()`.

Details

Each row is one draw. The columns `.chain`, `.iteration` and `.draw` identify the draw (the layout used by the **posterior** package), followed by one column per scalar parameter element: a vector parameter `b` gives `b[1]`, `b[2]`, ..., a matrix parameter `S` gives `S[1,1]`, `S[2,1]`, ...

Note that these names follow the R (1-based) convention used by **posterior**, **brms** and **bayesplot**, while indexing *inside model code* is passed to Python and is 0-based: `a[0]` in the model is column `a[1]` here.

Value

A `data.frame` (or a `posterior::draws_df`).

Examples

```
## Not run:
m <- importBF()
# ... m$fit(model) ...
draws <- bf_draws(m)
colMeans(draws[, -(1:3)])
posterior::summarise_draws(bf_draws(m, "draws_df"))
bayesplot::mcmc_trace(as.array(m))   # iterations x chains x parameters
long <- bf_draws(m, "long", labels = list(alpha = levels(df$group)))

## End(Not run)
```

---

bf_model — *Write a BayesForge model with automatic parameter names*

Description

Wraps an R model function so that every parameter assigned from a distribution gets the name of its variable, as in Python and in Julia (`@BF`): `a <- bf.dist.normal(0, 1)` becomes `a <- bf.dist.normal(0, 1, name = "a")`. An explicit `name =` is kept. Observed sites (`obs = y`) are already named after their variable by the distribution functions.

Usage

```
bf_model(f)
```

Arguments

f	A model function.

Details

The model body is rewritten once, when `bf_model()` is called; the function is otherwise unchanged. Both `<-` and `=` assignments are handled, also inside `with(m$dist$plate(...), ...)` blocks. The distribution calls recognised are `bf.dist.*(...)` and `m$dist$*(...)`.

Value

The same function, with `name =` added to its distribution calls.

Examples

```
library(BayesForge)
m <- importBF(platform = "cpu")
model <- bf_model(function(x, y) {
  a <- bf.dist.normal(0, 1)        # name "a"
  b <- bf.dist.normal(0, 1)        # name "b"
  s <- bf.dist.exponential(1)      # name "s"
  bf.dist.normal(a + b * x, s, obs = y)
})
```

---

bf_plot_ns *Wrap a BayesForge plot namespace so every method auto-displays plotly figures.*

Description

Wrap a BayesForge plot namespace so every method auto-displays plotly figures.

Usage

```
bf_plot_ns(py_ns)
```

Arguments

py_ns	A Python object (e.g. `m$diag`, `m$ppc`).

Value

An environment of class `"bf_plot_ns"` where calling any method automatically saves the returned plotly Figure to a temp HTML file and opens it in the browser.

bf_sample *Posterior / prior predictions as R arrays*

Description

Calls `m$sample()` and converts the returned JAX arrays into plain R objects, so the result can be used with any R function (`colMeans()`, `apply()`, matrix arithmetic, ...).

Usage

```
bf_sample(m, ...)
```

Arguments

`m`	A BayesForge object (from `importBF()`).
`...`	Passed on to `m$sample()`, e.g. `data =`, `samples =`, `seed =`, `remove_obs =`.

Details

`m$sample()` returns a named list whose values are JAX arrays (Python objects); R functions cannot work with those. `bf_sample()` keeps the names and turns every value into an R array: a `(draws, observations)` prediction becomes an R matrix, a 1-D result stays a plain numeric vector.

The arrays returned by `m$sample()` itself are left untouched – model code must keep receiving JAX arrays. Use `as.array()`, `as.matrix()` or `as.vector()` on a single JAX array when only that one is needed.

Value

A named list of R arrays (or a single R array when `m$sample()` returns one array instead of a list).

Examples

```
## Not run:
m <- importBF()
# ... m$fit(model) ...
pp <- bf_sample(m)
dim(pp$height)      # 4000 x 346
colMeans(pp$height)

## End(Not run)
```

bf_show *Display a BayesForge plotly figure in the browser.*

Description

Saves the figure to a temporary HTML file and opens it.

Usage

```
bf_show(fig)
```

Arguments

fig	A plotly Figure returned by a BayesForge diagnostic function.

Value

The figure, invisibly.

importBF *Start a BayesForge session*

Description

Binds R to the BayesForge Python environment (created once with `setup_env()`), imports the `BayesForge` Python module through **reticulate** and returns an initialised BayesForge object.

Usage

```
importBF(
  platform = "cpu",
  cores = NULL,
  rand_seed = TRUE,
  deallocate = FALSE,
  print_devices_found = TRUE,
  backend = "numpyro"
)
```

Arguments

platform	Character string, the computational platform to use (e.g. `"cpu"` or `"gpu"`). Defaults to `"cpu"`.
cores	Integer or `NULL`. Number of CPU cores to use. Defaults to `NULL`.
rand_seed	Logical or integer. An integer gives reproducible results; `TRUE` draws an entropy-based seed. Defaults to `TRUE`.

deallocate	Logical. Whether memory should be deallocated when not in use. Defaults to FALSE.
print_devices_found	Logical. Whether to print devices found. Defaults to TRUE.
backend	Character. Backend to use ("numpyro" or "tfp"). Defaults to "numpyro".

Details

- If the "BFR" Python environment does not exist yet, interactive sessions are offered to create it with setup_env(); otherwise an error explains how to do it. An existing environment is never reinstalled.
- jnp (jax.numpy) and jax are available as soon as the package is loaded, e.g. jnp$array(c(1, 2, 3)).

Value

An initialised BayesForge object (Python object via **reticulate**).

Examples

```
## Not run:
library(BayesForge)
m <- importBF()

## End(Not run)
```

---

jax_modules *JAX modules*

Description

jnp (jax.numpy) and jax, imported lazily from the BayesForge Python environment, for building arrays passed to BayesForge models, e.g. jnp$array(c(1, 2, 3)) or jnp$eye(3L).

---

loo.BayesForge.Main.main.bf *PSIS-LOO for a fitted BayesForge model*

Description

S3 method for loo::loo() on a fitted BayesForge object (from importBF() followed by m$fit()). The pointwise log-likelihood (iterations × chains × observations) is taken from BayesForge through ArviZ, then passed to loo::loo.array() with MCMC-based relative effective sample sizes, exactly as loo::loo_compare() expects.

Usage

```
loo.BayesForge.Main.main.bf(x, ...)
```

Arguments

x	A fitted BayesForge object.
...	Passed on to loo::loo.array() (e.g. cores, save_psis).

Details

R and Python agree up to PSIS smoothing details: on a linear regression, elpd_loo here matches m$diag$loo() in Python within 0.1.

Value

A "loo" object (see loo::loo.array()), usable with loo::loo_compare().

Examples

```
## Not run:
library(BayesForge)
m <- importBF()
# ... m$fit(model) ...
loo::loo(m)
loo::loo_compare(loo::loo(m1), loo::loo(m2))

## End(Not run)
```

---

setup_env *Create (or upgrade) the Python environment used by BayesForge*

Description

Creates a dedicated Python virtual environment (Python >= 3.12) and installs the bayesforge Python package into it, with JAX for the chosen backend. Versions of JAX, NumPyro and ArviZ are resolved by pip from the requirements of bayesforge itself, so the environment always matches the installed BayesForge release. Running it again upgrades an existing environment.

Usage

```
setup_env(env_name = "BFR", backend = c("cpu", "gpu"), python = NULL)
```

Arguments

env_name	Name of the virtual environment. Defaults to "BFR".
backend	Either "cpu" or "gpu" (Linux/WSL2 with an NVIDIA GPU).
python	Optional path to the Python interpreter used to create the environment. By default a Python >= 3.12 is located automatically.

Value

The path to the Python interpreter of the environment (invisibly).

Examples

```
## Not run:
setup_env()                # CPU
setup_env(backend = "gpu") # NVIDIA GPU, Linux/WSL2 only

## End(Not run)
```

`to_brms` *Convert BF fit to brmsfit*

Description

Converts a BF (NumPyro MCMC) object into a compatible brmsfit object.

Usage

```
to_brms(
  bi_fit,
  formula = NULL,
  data = NULL,
  family = NULL,
  par_map = NULL,
  ...
)
```

Arguments

`bi_fit`	The reticulate object returned by BF fitting.
`formula`	The brms formula for the model. Defaults to `bi_fit$formula`.
`data`	The data used for fitting. Defaults to `bi_fit$data`.
`family`	The family used. Defaults to `bi_fit$family` or `gaussian()`.
`par_map`	Optional list for custom naming: list(BI_name = brms_name).
`...`	Additional arguments passed to brms::brm(..., empty = TRUE).

Value

A brmsfit object.

Index

as.array(), *165*
as.data.frame(), *163*
as.data.frame.BayesForge.Main.main.bf *(*bf_draws*)*, 162
as.matrix(), *165*
as.vector(), *165*

bf.dist.asymmetric_laplace, 3
bf.dist.asymmetric_laplace_quantile, 5
bf.dist.bernoulli, 7
bf.dist.beta, 9
bf.dist.beta_binomial, 11
bf.dist.beta_proportion, 14
bf.dist.bfnomial *(*bf.dist.binomial*)*, 16
bf.dist.binomial, 16
bf.dist.car, 18
bf.dist.categorical, 21
bf.dist.cauchy, 23
bf.dist.chi2, 25
bf.dist.delta, 27
bf.dist.dirichlet, 29
bf.dist.dirichlet_multinomial, 31
bf.dist.discrete_uniform, 33
bf.dist.euler_maruyama, 35
bf.dist.exponential, 37
bf.dist.gamma, 39
bf.dist.gamma_poisson, 41
bf.dist.gaussian_copula, 43
bf.dist.gaussian_copula_beta, 45
bf.dist.gaussian_random_walk, 47
bf.dist.gaussian_state_space, 48
bf.dist.geometric, 50
bf.dist.gompertz, 52
bf.dist.gumbel, 54
bf.dist.half_cauchy, 56
bf.dist.half_normal, 58
bf.dist.inverse_gamma, 59
bf.dist.kumaraswamy, 61
bf.dist.laplace, 63
bf.dist.left_truncated_distribution, 65
bf.dist.levy, 67
bf.dist.lkj, 69
bf.dist.lkj_cholesky, 71
bf.dist.log_normal, 75
bf.dist.log_uniform, 77
bf.dist.logistic, 73
bf.dist.low_rank_multivariate_normal, 81
bf.dist.lower_truncated_power_law, 79
bf.dist.matrix_normal, 83
bf.dist.mixture, 86
bf.dist.mixture_general, 88
bf.dist.mixture_same_family, 91
bf.dist.multinomial, 93
bf.dist.multinomial_logits, 96
bf.dist.multinomial_probs, 98
bf.dist.multivariate_normal, 100
bf.dist.multivariate_student_t, 102
bf.dist.negative_binomial, 105
bf.dist.negative_binomial2, 107
bf.dist.negative_binomial_logits, 109
bf.dist.negative_binomial_probs, 111
bf.dist.normal, 112
bf.dist.ordered_logistic, 114
bf.dist.pareto, 117
bf.dist.poisson, 119
bf.dist.projected_normal, 121
bf.dist.relaxed_bernoulli, 122
bf.dist.relaxed_bernoulli_logits, 125
bf.dist.right_truncated_distribution, 127
bf.dist.soft_laplace, 129
bf.dist.student_t, 131
bf.dist.truncated_cauchy, 133
bf.dist.truncated_distribution, 135
bf.dist.truncated_normal, 138
bf.dist.truncated_polya_gamma, 140

bf.dist.two_sided_truncated_distribution, 142
bf.dist.uniform, 144
bf.dist.weibull, 146
bf.dist.wishart, 148
bf.dist.wishart_cholesky, 150
bf.dist.zero_inflated_distribution, 152
bf.dist.zero_inflated_negative_binomial, 155
bf.dist.zero_inflated_negative_binomial2 (bf.dist.zero_inflated_negative_binomial), 155
bf.dist.zero_inflated_poisson, 157
bf.dist.zero_sum_normal, 159
bf.doc, 161
bf_available, 162
bf_draws, 162
bf_model, 163
bf_model(), *4*, *6*, *8*, *10*, *12*, *14*, *17*, *19*, *22*, *24*, *26, 27*, *29*, *32*, *34*, *36*, *38*, *40, 41*, *43*, *45*, *47*, *49*, *51*, *53*, *55*, *57, 58*, *60*, *62*, *64*, *66*, *68*, *70*, *72*, *74*, *76*, *78*, *80*, *82*, *84*, *87*, *90*, *92*, *94*, *97*, *99*, *101*, *103*, *105*, *108, 109*, *111*, *113*, *115*, *117*, *120, 121*, *123*, *126*, *128*, *130*, *132*, *134*, *136*, *139*, *141, 142*, *144*, *146*, *149*, *151*, *153*, *156*, *158*, *160*
bf_plot_ns, 164
bf_sample, 165
bf_show, 166

importBF, 166
importBF(), *162*, *165*, *167*

jax (jax_modules), 167
jax_modules, 167
jnp (jax_modules), 167

loo.BayesForge.Main.main.bf, 167
loo::loo(), *167*
loo::loo.array(), *167, 168*
loo::loo_compare(), *168*

setup_env, 168
setup_env(), *162*, *166, 167*

to_brms, 169