Distributions

`Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

Asymmetric Laplace

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).

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

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

Args:

```
bi.dist.asymmetric_laplace(
loc=0.0,
scale=1.0,
asymmetry=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
```

```
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* (jnp.ndarray or float): Location parameter of the distribution.
- *scale* (jnp.ndarray or float): Scale parameter of the distribution.
- *asymmetry* (jnp.ndarray or float): Asymmetry parameter of the distribution.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.
- validate_args (bool, optional): Whether to enable validation of distribution parameters. Defaults to `None`.

Returns:

- When `sample=False`: A BI 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 BI AsymmetricLaplace distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.asymmetric_laplace(loc=0.0, scale=1.0, asymmetry=1.0, sample=True)
```

Wrapper of:

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

References

[1] "Asymmetric Laplace distribution", Wikipedia https://en.wikipedia.org/wiki/Asymmetric_Laplace_distribu

[2] Kozubowski TJ and Podgórski K. A Multivariate and Asymmetric Generalization of Laplace Distribution, Computational Statistics 15, 531–540 (2000). :doi:10.1007/PL00022717

–

Asymmetric Laplace Quantile

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.

Args:

```
bi.dist.asymmetric_laplace_quantile(
loc=0.0,
scale=1.0,
quantile=0.5,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
```

```
create_obj=False,
to_jax=True,
)
```

- *loc* (float): The location parameter of the distribution.
- *sample* (float): The scale parameter of the distribution.
- *quantile* (float): The quantile parameter, representing the proportion of probability density to the left of the median. Must be between 0 and 1.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI AsymmetricLaplaceQuantile distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.asymmetric_laplace_quantile(loc=0.0, scale=1.0, quantile=0.5, sample=True)
```

Wrapper of:

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

---

Bernoulli

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

$$X \sim Bernoulli(p)$$

with $0 \leq p \leq 1$.

Probability Mass Function (PMF)

$$P(X = x) = \Big\{p, \text{if } x = 1,\ 1 - p, \text{if } x = 0,\ 0, \text{otherwise.}$$

Alternatively, using an "indicator" exponent form:

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

Args:

```
bi.dist.bernoulli(
probs=None,
logits=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *probs* (jnp.ndarray, optional): Probability of success for each Bernoulli trial. Must be between 0 and 1.
- *logits* (jnp.ndarray, optional): Log-odds of success for each Bernoulli trial. `probs = sigmoid(logits)`.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.

- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI 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 BI distribution object (for advanced use cases) when `create_obj=True`.

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.bernoulli(probs=0.7, sample=True)
```

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

References:

- https://en.wikipedia.org/wiki/Bernoulli_distribution "Bernoulli distribution"
- https://stats.libretexts.org/Courses/Saint_Mary%27s_College_Notre_Dame/MATH_345___-*Probability*%28Kuter%29/3%3A_Discrete_Random_Variables/3.3%3A_Bernoulli_and_Binomial_Distr "3.3: Bernoulli and Binomial Distributions - Statistics LibreTexts"

Bernoulli Logits

Samples from a Bernoulli distribution parameterized by logits.

The Bernoulli distribution models a single binary event (success or failure), parameterized by the log-odds ratio of success. The probability of success is given by the sigmoid function applied to the logit.

$$p = \sigma(\eta) = \frac{1}{1 + e^{-\eta}}$$

where ( ) is the log-odds (the *logit*).

Args:

```
bi.dist.bernoulli_logits(
logits=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *logits* (jnp.ndarray, optional): Log-odds ratio of success. Must be real-valued.
- *shape* (tuple): 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.
- *event* (int, optional): The number of batch dimensions to reinterpret as *event* dimensions (used in model building). Defaults to 0.
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations. Defaults to None.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI BernoulliLogits distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the BernoulliLogits distribution (for direct sampling).
- When `create_obj=True`: The raw BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.bernoulli_logits(logits=jnp.array([0.2, 1, 2]), sample=True)
```

Wrapper of:

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

References

- https://en.wikipedia.org/wiki/Bernoulli_distribution "Bernoulli distribution"
- https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/Bernoulli "tfp. distributions.Bernoulli - Probability"
- https://mc-stan.org/docs/2_19/functions-reference/bernoulli-logit-distribution.html "11.2 Bernoulli Distribution, Logit Parameterization - Stan"

Bernoulli Probs

Samples from a Bernoulli distribution parameterized by probabilities.

The Bernoulli distribution models the probability of success in a single trial, where the outcome is binary (success or failure). It is characterized by a single parameter, `probs`, representing the probability of success.

$$Pr(X = 1) = p = 1 - Pr(X = 0) = 1 - q, f(x) = p \text{ if } k = 1 \text{ else } q \text{ if } k = 0$$

where: p is the probability of success (0 <= p <= 1)

Args:

```
bi.dist.bernoulli_probs(
probs,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *probs* (jnp.ndarray): The probability of success for each Bernoulli trial. Must be between 0 and 1.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI BernoulliProbs distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the BernoulliLogits distribution (for direct sampling).
- When `create_obj=True`: The raw BI BernoulliProbs object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.bernoulli_probs(probs=jnp.array([0.2, 0.7, 0.5]), sample=True)
```

Wrapper of:

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

---

Beta

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

Args:

```
bi.dist.beta(
concentration1,
concentration0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *concentration1* (jnp.ndarray): The first concentration parameter (shape parameter). Must be positive.
- *concentration0* (jnp.ndarray): The second concentration parameter (shape parameter). Must be positive.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.beta(concentration1=1.0, concentration0=1.0, sample=True)
```

Wrapper of:

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

References:

- https://en.wikipedia.org/wiki/Beta_distribution "Beta distribution - Wikipedia"
- https://www.statlect.com/probability-distributions/beta-distribution "Beta distribution | Properties, proofs, exercises - StatLect"
- https://reference.wolfram.com/language/ref/BetaDistribution.html "BetaDistribution - Wolfram Language Documentation"

BetaBinomial

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 $X \sim BetaBinomial(N, \alpha, \beta)$, 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 $(B(u, v) = \frac{\Gamma(u),\Gamma(v)}{\Gamma(u+v)})$.

Args:

```
bi.dist.beta_binomial(
concentration1,
concentration0,
total_count=1,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *concentration1* (jnp.ndarray): The first concentration parameter (alpha) of the Beta distribution.
- *concentration0* (jnp.ndarray): The second concentration parameter (beta) of the Beta distribution.
- *total_count* (jnp.ndarray): The number of Bernoulli trials in the Binomial part of the distribution.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.

- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI BetaBinomial distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.beta_binomial(concentration1=1.0, concentration0=1.0, total_count=10, sample=True)
```

Wrapper of:

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

References:

- https://en.wikipedia.org/wiki/Beta-binomial_distribution "Beta-binomial distribution"
- https://www.statisticshowto.com/beta-binomial-distribution/ "Beta-Binomial Distribution: Definition - Statistics How To"
- https://mc-stan.org/docs/2_19/functions-reference/beta-binomial-distribution.html "12.3 Beta-Binomial Distribution | Stan Functions Reference"

Beta Proportion

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((,)), 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 ( ) (the "center") and ( ) (the "spread").

Args:

```
bi.dist.beta_proportion(
mean,
concentration,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *mean* (jnp.ndarray): The mean of the BetaProportion distribution,must be between 0 and 1.
- *concentration* (jnp.ndarray): The concentration parameter of the BetaProportion distribution.
- *shape* (tuple): 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.

- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI BetaProportion distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
samples = m.dist.beta_proportion(mean=0.5, concentration=2.0, sample=True, shape=(1000,))
```

Wrapper of:

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

References:

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

Binomial

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:

$$X \sim Binomial(n, p).$$

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 $p^k(1-p)^{n-k}$.

Args:

```
bi.dist.binomial(
total_count=1,
probs=None,
logits=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *total_count* (int): The number of trials n.
- *probs* (jnp.ndarray, optional): The probability of success p for each trial. Must be between 0 and 1.
- *logits* (jnp.ndarray, optional): The log-odds of success for each trial. `probs = jax.nn.sigmoid(logits)`.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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 BI distribution object (for advanced use cases) when `create_obj=True`.

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.binomial(total_count=10, probs=0.5, sample=True)
```

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

References:

- https://en.wikipedia.org/wiki/Binomial_distribution "Binomial distribution"
- https://www.itl.nist.gov/div898/handbook/eda/section3/eda366i.htm "1.3.6.6.18. Binomial Distribution - Information Technology Laboratory"

Binomial Logits

The BinomialLogits distribution represents a binomial distribution parameterized by logits. It is useful when the probability of success is not directly known but is instead expressed as logits, which are the natural logarithm of the odds ratio.

$$p = logit^{-1}(\alpha) = \frac{1}{1 + e^{-\alpha}}.$$

Args:

```
bi.dist.binomial_logits(
logits,
total_count=1,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *logits* (jnp.ndarray): Log-odds of each success.
- *total_count* (int): Number of trials.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI BinomialLogits distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the BernoulliLogits distribution (for direct sampling).
- When `create_obj=True`: The raw BI BinomialLogits object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.binomial_logits(logits=jnp.zeros(10), total_count=5, sample=True)
```

Wrapper of:

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

---

Binomial Probs

Samples from a Binomial distribution with specified probabilities for each trial.

The Binomial distribution models the number of successes in a sequence of independent Bernoulli trials, where each trial has the same probability of success.

For $k = 0, 1, 2, \ldots, N$:

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

where $\binom{N}{k} = \frac{N!}{k!(N-k)!}$.

Args:

```
bi.dist.binomial_probs(
probs,
total_count=1,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *probs* (jnp.ndarray): The probability of success for each trial. Must be between 0 and 1.
- *total_count* (int): The number of trials in each sequence.
- *shape* (tuple): 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.

- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI BinomialProbs distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the BernoulliLogits distribution (for direct sampling).
- When `create_obj=True`: The raw BI BinomialLogits object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.binomial_probs(probs=0.5, total_count=10, sample=True)
```

Wrapper of:

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

---

Conditional Autoregressive (CAR)

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 $y = (y_1, y_2, \ldots, y_N)$ 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 y_{-i} follows a normal distribution:

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

where:

- $N(i)$ 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.

Args:

```
bi.dist.car(
loc,
correlation,
conditional_precision,
adj_matrix,
is_sparse=False,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
```

```
create_obj=False,
to_jax=True,
)
```

- *loc* (Union[float, Array]): Mean of the distribution.
- *correlation* (Union[float, Array]): Correlation between variables.
- *conditional_precision* (Union[float, Array]): Precision of the distribution.
- *adj_matrix* (Union[Array, scipy.sparse.spmatrix]): 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 None.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI CAR distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the BernoulliLogits distribution (for direct sampling).
- When `create_obj=True`: The raw BI CAR object (for advanced use cases).

References:

- https://www.pymc.io/projects/examples/en/latest/spatial/conditional_autoregressive_priors.html "Conditional Autoregressive (CAR) Models for Spatial Data"
- https://mc-stan.org/learn-stan/case-studies/icar_stan.html "Intrinsic Auto-Regressive Models for Areal Data - Stan"
- https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat08048 "Conditional Autoregressive (CAR) Model - Schmidt"

Categorical distribution.

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 0) for all (i) and $ _{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.

Args:

```
bi.dist.categorical(
probs=None,
logits=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *probs* (jnp.ndarray): A 1D array of probabilities for each category. Must sum to 1.
- *logits* (jnp.ndarray): A 1D array of unnormalized log probabilities for each category.
- *shape* (tuple): A multi-purpose argument for shaping. When `sample=False` (model building), this is used with `.expand(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.

- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.categorical(probs=jnp.array([0.2, 0.3, 0.5]), sample=True)
```

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

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_distribut distribution - Wikipedia”

Categorical Logits

Samples from a Categorical distribution with logits. This distribution represents a discrete probability distribution over a finite set of outcomes, where the probabilities are determined by the logits. The probability of each outcome is given by the softmax function applied to the logits.

Given logits $ = (_1, ..., _K) $, the probability of each category $ i $ is:

$$p_i = \frac{e^{\gamma_i}}{\sum_{j=1}^{K} e^{\gamma_j}}$$

where $ p_i is the probability of category $ i , and the sum in the denominator ensures that the probabilities sum to 1.

Args:

```
bi.dist.categorical_logits(
logits,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *logits* (jnp.ndarray): Log-odds of each category.

- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI CategoricalLogits distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the BernoulliLogits distribution (for direct sampling).
- When `create_obj=True`: The raw BI CategoricalLogits object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.categorical_logits(logits=jnp.zeros(5), sample=True)
```

Wrapper of:

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

---

Categorical Probs distribution.

Samples from a Categorical distribution.

The Categorical distribution is a discrete probability distribution that represents the probability of each outcome from a finite set of possibilities. It is often used to model the outcome of a random process with a fixed number of possible outcomes, such as the roll of a die or the selection of an item from a list.

$$P(x) = \frac{probs_i}{\sum_{k=1}^{K} probs_k}$$

Args:

```
bi.dist.categorical_probs(
probs,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *probs* (jnp.ndarray): Probabilities for each category. Must sum to 1.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.

- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI CategoricalProbs distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the BernoulliLogits distribution (for direct sampling).
- When `create_obj=True`: The raw BI CategoricalProbs object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.categorical_probs(probs=jnp.array([0.2, 0.3, 0.5]), sample=True)
```

Wrapper of:

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

---

Cauchy

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.

Args:

```
bi.dist.cauchy(
loc=0.0,
scale=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* (jnp.ndarray or float, optional): Location parameter. Defaults to 0.0.
- *sample* (jnp.ndarray or float, optional): Scale parameter. Must be positive. Defaults to 1.0.
- *shape* (tuple): 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.
- *event* (int, optional): The number of batch dimensions to reinterpret as event dimensions (used in model building). Defaults to None.
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations. Defaults to None.

- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.cauchy(loc=0.0, scale=1.0, sample=True)
```

Wrapper of:

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

Chi-squared

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.

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

where: * (x) is the random variable, * (k) is the degrees of freedom, * (( )) is the gamma function.

Args:

```
bi.dist.chi2(
df,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *df* (jnp.ndarray): Degrees of freedom. Must be positive.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.

- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI Chi2 distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the BernoulliLogits distribution (for direct sampling).
- When `create_obj=True`: The raw BI Chi2 object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu)
m.dist.chi2(df=3.0, sample = True)
```

Wrapper of:

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

Circulant Normal Multivariate normal

Circulant Normal Multivariate normal distribution with covariance matrix ${C}`that is positive-definite and circulant [1], i.e., has periodic boundary conditions. The density of a sample ${x}\in{R}^n` is the standard multivariate normal density

$$p\left(x \mid \mu, C\right) = \frac{\left(det\, C\right)^{-1/2}}{\left(2\pi\right)^{n/2}} \exp\left(-\frac{1}{2}\left(x-\mu\right)^{\top} C^{-1}\left(x-\mu\right)\right),$$

where *det* denotes the determinant and $^\top$ the transpose. Circulant matrices can be diagnolized efficiently using the discrete Fourier transform, allowing the log likelihood to be evaluated in $n \log n$ time for n observations.

- *loc:* Mean of the distribution μ.
- *covariance_row:* First row of the circulant covariance matrix C. Because of periodic boundary conditions, the covariance matrix is fully determined by its first row (see `jax.scipy.linalg.toeplitz` for further details).
- *covariance_rfft:* Real part of the real fast Fourier transform of :code:`covariance_row`, the first row of the circulant covariance matrix C.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

References:

1. Wikipedia. (n.d.). Circulant matrix. Retrieved March 6, 2025, from https://en.wikipedia.org/wiki/Circula
2. Wood, A. T. A., & Chan, G. (1994). Simulation of Stationary Gaussian Processes in $[0, 1]^d$. *Journal of Computational and Graphical Statistics*, 3(4), 409–432. https://doi.org/10.1080/10618600.1994.10474655

Args:

```
bi.dist.circulant_normal(
loc: jax.Array,
covariance_row: jax.Array = None,
covariance_rfft: jax.Array = None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
```

```
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* : jnp.ndarray Mean of the distribution μ.
- *covariance_row* : jnp.ndarray, optional. First row of the circulant covariance matrix C. Defaults to None.
- *covariance_rfft* : jnp.ndarray, optional Real part of the real fast Fourier transform of `covariance_row`. Defaults to None.

Returns:

- When `sample=False`: A BI Circulant Normal Distribution distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the BernoulliLogits distribution (for direct sampling).
- When `create_obj=True`: The raw BI Circulant Normal Distribution object (for advanced use cases).

Wrapper of: https://num.pyro.ai/en/stable/distributions.html#circulantnormal

References:

Wikipedia. (n.d.). Circulant matrix. Retrieved March 6, 2025, from https://en.wikipedia.org/wiki/Circulant_m

Wood, A. T. A., & Chan, G. (1994). Simulation of Stationary Gaussian Processes in $[0, 1]^d$.
Journal of Computational and Graphical Statistics, 3(4), 409–432. https://doi.org/10.1080/10618600.1994.10474

Delta

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.

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

Args:

```
bi.dist.delta(
v=0.0,
log_density=0.0,
event_dim=0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *v* (jnp.ndarray): The location of the point mass.
- *log_density* (float, optional): 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* (int, optional): The number of rightmost dimensions of `v` to interpret as event dimensions. Defaults to 0.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.

- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.delta(v=0.0, sample=True)
```

Wrapper of:

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

Dirichlet

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 $X = (X_1, \ldots, X_K \sim Dir(\alpha)$, 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):

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

Args:

```
bi.dist.dirichlet(
concentration,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *concentration* (jnp.ndarray): The concentration parameter(s) of the Dirichlet distribution. Must be a positive array.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.

- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI Dirichlet Distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.dirichlet(concentration=jnp.array([1.0, 1.0, 1.0]), sample=True)
```

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

References

https://distribution-explorer.github.io/multivariate_continuous/dirichlet.html https://en.wikipedia.org/wiki/D

Dirichlet-Multinomial

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

$$p = (p_1, \dots, p_K) \sim Dirichlet(\alpha)$$

and then you draw counts

$$x = (x_1, \dots, x_K) \sim Multinomial(n, p).$$

Integrating out p gives you the marginal distribution of ${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.

Args:

```
bi.dist.dirichlet_multinomial(
concentration,
total_count=1,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *concentration* (jnp.ndarray): The concentration parameter (alpha) for the Dirichlet prior. Values must be positive. The last dimension is interpreted as the number of categories.
- *total_count* (int, jnp.ndarray, optional): The total number of trials (n). This must be a non-negative integer. Defaults to 1.
- *validate_args* (bool, optional): Whether to enable validation of distribution parameters. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations. If provided, events with a `True` mask will be conditioned on `obs`, while the remaining events will be treated as latent variables. Defaults to `None`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.

- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`. Defaults to 0.
- *shape* (tuple, optional): A multi-purpose argument for shaping. When `sample=False` (model building), this is used with `.expand(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.
- *event* (int, optional): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to `False`.

Returns:

- When `sample=False`: A BI 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 BI dirichlet_multinomial Distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
import jax.numpy as jnp
m = bi('cpu')

# Direct sampling
# Sample a single vector of counts for 10 trials from 3 categories
counts = m.dist.dirichlet_multinomial(concentration=jnp.array([1.0, 1.0, 1.0]),total_count=1

# Usage within a model
def my_model(obs_data=None):
# Define a prior on the concentration parameter
alpha = m.dist.half_cauchy(scale=jnp.ones(5), name='alpha', shape=(5,))

# Model observed counts
with m.plate('data', len(obs_data)):
```

```
y = m.dist.dirichlet_multinomial(
concentration=alpha,
total_count=100,
name='y',
obs=obs_data
)
```

Wrapper of:

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

References:

https://en.wikipedia.org/wiki/Dirichlet-multinomial_distribution https://docs.scipy.org/doc/scipy/reference/g

---

Discrete Uniform

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 $.

Args:

```
bi.dist.discrete_uniform(
low=0,
high=1,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
```

```
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *low* (jnp.ndarray): The lower bound of the uniform range, inclusive.
- *high* (jnp.ndarray): The upper bound of the uniform range, inclusive.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI DiscreteUniform Distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.discrete_uniform(low=0, high=5, sample=True)
```

Wrapper of:

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

References:

Doubly Truncated Power Law

This distribution represents a continuous power law with a finite support bounded between `low` and `high`, and with an exponent `alpha`. It is normalized over the interval `[low, high]` to ensure the area under the density function is 1.

Key parameters:

- $\alpha > 0$: shape (exponent) of the power law
- x_{min}: lower cutoff (minimum possible value)
- $x_{\max}$: upper cutoff (maximum possible value)

The probability density function (PDF) is defined as:

$$f(x) = \begin{cases} \frac{(1-\alpha), x^{-\alpha}}{x_{\max}^{1-\alpha} - x_{min}^{1-\alpha}}, \alpha \neq 1, ; x_{min} \leq x \leq x_{\max}, \\ \frac{1}{x, \ln\left(x_{\max}/x_{min}\right)}, \alpha = 1, ; x_{min} \leq x \leq x_{\max}, \\ 0, \text{otherwise.} \end{cases}$$

where the normalization constant $Z(\alpha, a, b)$ is given by

$$Z(\alpha, a, b) = \begin{cases} \log(b) - \log(a), \text{if } \alpha = -1, \\ \frac{b^{1+\alpha} - a^{1+\alpha}}{1+\alpha}, \text{otherwise.} \end{cases}$$

This distribution is useful for modeling data that follows a power-law behavior but is naturally bounded due to measurement or theoretical constraints (e.g., finite-size systems).

Args:

```
bi.dist.doubly_truncated_power_law(
alpha,
low,
high,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *alpha* (float or array-like): Power-law exponent.
- *low* (float or array-like): Lower bound of the distribution (must be ≥ 0).
- *high* (float or array-like): Upper bound of the distribution (must be > 0).
- *shape* (tuple): 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 (bool, optional): Whether to validate the arguments. Defaults to True.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI doubly_truncated_power_law Distribution distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the doubly_truncated_power_law distribution (for direct sampling).
- When `create_obj=True`: The raw BI doubly_truncated_power_law Distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.doubly_truncated_power_law(low=0.1, high=10.0, alpha=2.0, sample=True)
```

References:

https://reference.wolframcloud.com/language/ref/TruncatedDistribution.html

---

Euler–Maruyama

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_tis 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).

Args:

```
bi.dist.euler_maruyama(
t,
sde_fn,
init_dist,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=0,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *t* (jnp.ndarray): Discretized time steps.
- *sde_fn* (callable): A function that takes the current state and time as input and returns the drift and diffusion coefficients.
- *init_dist* (Distribution): The initial distribution of the system.
- *shape* (tuple): 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.
- *sample_shape (tuple, optional): The shape of the samples to draw. Defaults to None.
- *validate_args (bool, optional): Whether to validate the arguments. Defaults to True.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

jnp.ndarray: Samples drawn from the Euler–Maruyama distribution.

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.euler_maruyama(t=jnp.array([0.0, 0.1, 0.2]), sde_fn=lambda x, t: (x, 1.0), init_dist=
```

Exponential

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 0 $

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

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

Args:

```
bi.dist.exponential(
rate=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *rate* (jnp.ndarray): The rate parameter, $ ‘. Must be positive.
- *shape* (tuple): 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.

- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.exponential(rate=1.0, sample=True)
```

Wrapper of:

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

References

https://reference.wolfram.com/language/ref/ExponentialDistribution.html https://en.wikipedia.org/wiki/Expo

---

Folded

Samples from a Folded distribution, which is the absolute value of a base univariate distribution. This distribution reflects the base distribution across the origin, effectively taking the absolute value of each sample.

$$p(x) = \sum_{k=-\infty}^{\infty} p(x - 2k)$$

Args:

```
bi.dist.folded_distribution(
base_dist,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* (float, optional): Location parameter of the base distribution. Defaults to 0.0.
- *sample* (float, optional): Scale parameter of the base distribution. Defaults to 1.0.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.

- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI FoldedDistribution distribution object (for model building) when `sample=False`. JAX array of samples drawn from the FoldedDistribution distribution (for direct sampling) when `sample=True`. The raw BI distribution object (for advanced use cases) when `create_obj=True`.

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.folded_distribution(m.dist.normal(loc=0.0, scale=1.0, create_obj = True), sample=True)
```

Wrapper of:

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

---

Gamma

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 (λ).

There are two common parameterizations:

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

Args:

```
bi.dist.gamma(
concentration,
rate=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *concentration* (jnp.ndarray): The shape parameter of the Gamma distribution (k $>$ 0).
- *rate* (jnp.ndarray): The rate parameter of the Gamma distribution (theta $>$ 0).
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

Gamma: A BI Gamma distribution object (for model building).

jnp.ndarray: A JAX array of samples drawn from the Gamma distribution (for direct sampling).

Gamma: The raw BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.gamma(concentration=2.0, rate=0.5, sample=True)
```

Wrapper of:

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

References

https://reference.wolfram.com/language/ref/GammaDistribution.html https://en.wikipedia.org/wiki/Gamma_

Gamma Poisson

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.

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

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.

Args:

```
bi.dist.gamma_poisson(
concentration,
rate=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *concentration* (jnp.ndarray): Shape parameter (alpha) of the Gamma distribution.
- *rate* (jnp.ndarray): Rate parameter (beta) for the Gamma distribution.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.gamma_poisson(concentration=1.0, rate=2.0, sample=True)
```

Wrapper of:

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

References

[1] https://www.statisticshowto.com/gamma-poisson-distribution https://en.wikipedia.org/wiki/Negative_binc

---

Gaussian Copula

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.

Args:

```
bi.dist.gaussian_copula(
marginal_dist,
correlation_matrix=None,
correlation_cholesky=None,
validate_args=None,
name='x',
obs=None,
```

```
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *marginal_dist* (Distribution): Distribution whose last batch axis is to be coupled.
- *correlation_matrix* (array_like, optional): Correlation matrix of the coupling multivariate normal distribution. Defaults to None.
- *correlation_cholesky* (array_like, optional): Correlation Cholesky factor of the coupling multivariate normal distribution. Defaults to None.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations. Defaults to None.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI GaussianCopula distribution object: When `sample=False` (for model building). JAX array: When `sample=True` (for direct sampling). BI distribution object: When `create_obj=True` (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.gaussian_copula(
marginal_dist = m.dist.beta(2.0, 5.0, create_obj = True),
correlation_matrix = jnp.array([[1.0, 0.7],[0.7, 1.0]]),
sample = True
)
```

Wrapper of:

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

---

Gaussian Copula Beta

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.

Args:

```
bi.dist.gaussian_copula_beta(
concentration1,
concentration0,
correlation_matrix=None,
correlation_cholesky=None,
validate_args=False,
name='x',
obs=None,
mask=None,
sample=False,
```

```
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *concentration1* (jnp.ndarray): The first shape parameter of the Beta distribution.
- *concentration0* (jnp.ndarray): The second shape parameter of the Beta distribution.
- *correlation_matrix* (array_like, optional): Correlation matrix of the coupling multivariate normal distribution. Defaults to None.
- *correlation_cholesky* (jnp.ndarray): The Cholesky decomposition of the correlation matrix.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

GaussianCopulaBeta: A BI GaussianCopulaBeta distribution object (for model building). jnp.ndarray: A JAX array of samples drawn from the GaussianCopulaBeta distribution (for direct sampling). Distribution: The raw BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.gaussian_copula_beta(
concentration1 = jnp.array([2.0, 3.0]),
concentration0 = jnp.array([5.0, 3.0]),
correlation_cholesky = jnp.linalg.cholesky(jnp.array([[1.0, 0.7],[0.7, 1.0]])),
sample = True
)
```

Wrapper of:

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

---

Gaussian Random Walk

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.

Args:

```
bi.dist.gaussian_random_walk(
scale=1.0,
num_steps=1,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *scale* (float, jnp.ndarray, optional): The standard deviation ($ ‘) 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 `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to ‘x’.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations. If provided, events with a `True` mask will be conditioned on `obs`, while the remaining events will be treated as latent variables. Defaults to `None`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`. Defaults to 0.
- *shape* (tuple, optional): A multi-purpose argument for shaping. When `sample=False` (model building), this is used with `.expand(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.
- *event* (int, optional): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to `False`.

Returns:

BI.primitives.Messenger: A BI sample site object when used in a model context (`sample=False`). jnp.ndarray: A JAX array of samples drawn from the GaussianRandomWalk distribution (for direct sampling, `sample=True`). numpyro.distributions.Distribution: The raw BI distribution object (if `create_obj=True`).

Example Usage:

```
from BI import bi
import jax.numpy as jnp
m = bi('cpu')

# Direct sampling of a random walk with 100 steps
path = m.dist.gaussian_random_walk(scale=0.5, num_steps=100, sample=True)

# Usage within a model for a latent time series
def my_model(data=None):
# Prior on the volatility of the random walk
volatility = m.dist.half_cauchy(scale=1.0, name='volatility')

# The latent random walk
latent_process = m.dist.gaussian_random_walk(
scale=volatility,
num_steps=len(data) if data is not None else 10,
name='latent_process'
)

# Observation model
# Assumes the observed data is the latent process plus some noise
obs_noise = m.dist.half_cauchy(scale=1.0, name='obs_noise')
with m.plate('time', len(data) if data is not None else 10):
return m.dist.normal(loc=latent_process, scale=obs_noise, obs=data, name='obs')
```

Wrapper of:

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

---

Gaussian State Space

Samples from a Gaussian state space model.

Args:

```
bi.dist.gaussian_state_space(
num_steps,
transition_matrix,
covariance_matrix=None,
precision_matrix=None,
scale_tril=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *num_steps* (int): Number of steps.
- *transition_matrix* (jnp.ndarray): State space transition matrix ${A}‘.
- *covariance_matrix* (jnp.ndarray, optional): Covariance of the innovation noise ϵ. Defaults to None.
- *precision_matrix (jnp.ndarray, optional): Precision matrix of the innovation noise ϵ. Defaults to None.
- *scale_tril (jnp.ndarray, optional): Scale matrix of the innovation noise ϵ. Defaults to None.
- *shape* (tuple): A multi-purpose argument for shaping. When `sample=False` (model building), this is used with `.expand(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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations. Defaults to None.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. Defaults to False.

- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.gaussian_state_space(num_steps=5, transition_matrix=jnp.array([[0.5]]), covariance_ma
```

Wrapper of:

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

---

Geometric distribution.

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 $1, 2, 3, ...$).

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

Args:

```
bi.dist.geometric(
probs=None,
logits=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *probs* (jnp.ndarray, optional): Probability of success on each trial. Must be between 0 and 1.
- *logits* (jnp.ndarray, optional): Log-odds of success on each trial. `probs = jax.nn.sigmoid(logits)`.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.

- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.geometric(probs=0.5, sample=True)
```

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

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"

GeometricLogits

Samples from a GeometricLogits distribution, which models the number of failures before the first success in a sequence of independent Bernoulli trials. It is parameterized by logits, which are transformed into probabilities using the sigmoid function.

$$P(X = k) = (1 - p)^k, p,$$

where:

- X is the number of failures before the first success.
- k is the number of failures.
- p is the probability of success on each trial (derived from the logits).

Args:

```
bi.dist.geometric_logits(
logits,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *logits* (jnp.ndarray): Log-odds parameterization of the probability of success.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI GeometricLogits distribution object (for model building) when `sample=False`. JAX array of samples drawn from the GeometricLogits distribution (for direct sampling) when `sample=True`. The raw BI distribution object (for advanced use cases) when `create_obj=True`.

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.geometric_logits(logits=jnp.zeros(10), sample=True)
```

Wrapper of:

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

References:

https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/Geometric

---

GeometricProbs

Samples from a Geometric

The Geometric distribution models the number of trials until the first success in a sequence of independent Bernoulli trials, where each trial has the same probability of success.

$$P(X = k) = (1 - p)^k p$$

Args:

```
bi.dist.geometric_probs(
probs,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
```

```
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *probs* (jnp.ndarray): Probability of success on each trial. Must be between 0 and 1.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.geometric_probs(probs=0.5, sample=True)
```

Wrapper of:

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

---

Gompertz

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:

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

Args:

```
bi.dist.gompertz(
concentration,
rate=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *concentration* (jnp.ndarray): The concentration parameter. Must be positive.
- *rate* (jnp.ndarray): The rate parameter. Must be positive.
- *shape* (tuple): 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.

- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI Gompertz distribution object: When `sample=False` (for model building). JAX array: When `sample=True` (for direct sampling). BI distribution object: When `create_obj=True` (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.gompertz(concentration=1.0, rate=1.0, sample=True)
```

Wrapper of:

https://num.pyro.ai/en/stable/distributions.html#gompertz #### References: https://en.wikipedia.org/wiki

---

Gumbel

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.

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

Args:

```
bi.dist.gumbel(
loc=0.0,
scale=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* (jnp.ndarray or float, optional): Location parameter. Defaults to 0.0.
- *scale* (jnp.ndarray or float, optional): Scale parameter. Must be positive. Defaults to 1.0.
- *shape* (tuple): 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.
- *event* (int, optional): The number of batch dimensions to reinterpret as event dimensions (used in model building). Defaults to None.
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations. Defaults to None.

- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.gumbel(loc=0.0, scale=1.0, sample=True)
```

Wrapper of:

https://num.pyro.ai/en/stable/distributions.html#gumbel #### References: https://en.wikipedia.org/wiki/G

HalfCauchy

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.

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

Args:

```
bi.dist.half_cauchy(
scale=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *scale* (jnp.ndarray): The scale parameter of the Cauchy distribution. Must be positive.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.half_cauchy(scale=1.0, sample=True)
```

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

---

HalfNormal

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.

$$f(x;\sigma) = \frac{2}{\pi,\sigma};\frac{1}{1+\left(x/\sigma\right)^{2}}, \qquad x \geq 0.$$

Args:

```
bi.dist.half_normal(
scale=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
```

```
to_jax=True,
)
```

- *scale* (float, array): The scale parameter of the distribution. Must be positive.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.half_normal(scale=1.0, sample=True)
```

Wrapper of:

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

References:

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

---

Improper Uniform

A helper distribution with zero :meth:`log_prob` over the `support` domain.

Args:

```
bi.dist.improper_uniform(
support,
batch_shape,
event_shape,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *support* (numpyro.distributions.constraints.Constraint): The support of this distribution.
- *batch_shape* (tuple): Batch shape of this distribution. It is usually safe to set `batch_shape=()`.
- *event_shape* (tuple): Event shape of this distribution.
- *shape* (tuple): A multi-purpose argument for shaping. When `sample=False` (model building), this is used with `.expand(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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.

- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI ImproperUniform distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the ImproperUniform distribution (for direct sampling).
- When `create_obj=True`: The raw BI distribution object (for advanced use cases).

Example Usage:

```
from BI import sample
from numpyro.distributions import ImproperUniform, Normal, constraints

def model():
x = sample('x', ImproperUniform(constraints.ordered_vector, (), event_shape=(10,)))
```

Wrapper of:

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

---

Inverse Gamma

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: * (( )) is the Gamma function.

Args:

```
bi.dist.inverse_gamma(
concentration,
rate=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *concentration* (jnp.ndarray): The concentration parameter α of the InverseGamma distribution. Must be positive.
- *rate* (jnp.ndarray): The rate parameter β of the InverseGamma distribution. Must be positive.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.

- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.inverse_gamma(concentration=2.0, rate=1.0, sample=True)
```

Wrapper of:

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

---

Kumaraswamy

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.

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

where: * (a) and (b) are shape parameters. * (x) is the random variable.

Args:

```
bi.dist.kumaraswamy(
concentration1,
concentration0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *concentration1* (jnp.ndarray): The first shape parameter. Must be positive.
- *concentration0* (jnp.ndarray): The second shape parameter. Must be positive.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.kumaraswamy(concentration1=2.0, concentration0=3.0, sample=True)
```

Wrapper of:

https://num.pyro.ai/en/stable/distributions.html#kumaraswamy #### Rerferences: https://en.wikipedia.org/wiki/Kumaraswamy_distribution

---

Laplace

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.

Args:

```
bi.dist.laplace(
loc=0.0,
scale=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
```

```
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* (jnp.ndarray): Location parameter of the Laplace distribution.
- *sample* (jnp.ndarray): Scale parameter of the Laplace distribution. Must be positive.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.laplace(loc=0.0, scale=1.0, sample=True)
```

Wrapper of:

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

Left Truncated

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).

Args:

```
bi.dist.left_truncated_distribution(
base_dist,
low=0.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *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.

- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

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

References:

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

---

Levy

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

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

Args:

```
bi.dist.levy(
loc,
scale,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* (jnp.ndarray): Location parameter.
- *sample* (jnp.ndarray): Scale parameter.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.

- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI Levy distribution object: When `sample=False` (for model building). JAX array: When `sample=True` (for direct sampling). BI distribution object: When `create_obj=True` (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.levy(loc=0.0, scale=1.0, sample=True)
```

Wrapper of:

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

Lewandowski Kurowicka Joe (LKJ)

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.

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

Args:

```
bi.dist.lkj(
dimension,
concentration=1.0,
sample_method='onion',
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *dimension* (int): The dimension of the correlation matrices.
- *concentration* (ndarray): The concentration/shape parameter of the distribution (often referred to as eta). Must be positive.
- *sample_method* (str): 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".
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).

- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.lkj(dimension=2, concentration=1.0, sample=True)
```

Wrapper of:

https://num.pyro.ai/en/stable/distributions.html#lkj #### References: https://en.wikipedia.org/wiki/Kuma

---

LKJ Cholesky

The LKJ (Leonard-Kjærgaard-Jørgensen) 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 Bayesian inference 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). * η 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.

Args:

```
bi.dist.lkj_cholesky(
dimension,
concentration=1.0,
sample_method='onion',
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *dimension (int): The dimension of the correlation matrices.
- *concentration (float): A parameter controlling the concentration of the distribution around the identity matrix. Higher values indicate greater concentration. Must be greater than 1.

- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Attributes: concentration (float): The concentration parameter.

References:

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

---

Log-Normal

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:

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

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).

Parameters

- μ: the mean of $\ln X$ (i.e. the location parameter in log-space)
- $\sigma > 0$: the standard deviation of $\ln X$ (i.e. the "scale" in log-space)

Sometimes another parameterization uses a threshold (shift) θ, i.e. $X = \theta + \exp(Y)$. For the two-parameter (no shift) case:

- **PDF**

$$f(x;\mu,\sigma) = \frac{1}{x,\sigma\sqrt{2\pi}};\exp!\Big(-\frac{(\ln x-\mu)^2}{2\sigma^2}\Big), \quad x>0.$$

Args:

```
bi.dist.log_normal(
loc=0.0,
scale=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* (float): Location parameter.
- *scale* (float): Scale parameter.
- *shape* (tuple): 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.

- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.log_normal(loc=0.0, scale=1.0, sample=True)
```

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

References:

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

Log-Uniform

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 $x > b$, $F(x) = 1$.

Args:

```
bi.dist.log_uniform(
low,
high,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *low* (jnp.ndarray): The lower bound of the uniform distribution's log-space. Must be positive.
- *high* (jnp.ndarray): The upper bound of the uniform distribution's log-space. Must be positive.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI 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 BI distribution object (for advanced use cases) when `create_obj=True`.

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.log_uniform(low=0.1, high=10.0, sample=True)
```

Wrapper of:

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

References:

https://en.wikipedia.org/wiki/Reciprocal_distribution https://docs.scipy.org/doc/scipy/reference/generated/s
https://docs.scipy.org/doc/scipy/tutorial/stats/continuous_loguniform.html

---

Logistic

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

Let $X \sim Logistic(\mu, s)$. Then:

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

Equivalently:

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

Args:

```
bi.dist.logistic(
loc=0.0,
scale=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* (jnp.ndarray or float): The location parameter, specifying the median of the distribution. Defaults to 0.0.
- *scale* (jnp.ndarray or float): The scale parameter, which determines the spread of the distribution. Must be positive. Defaults to 1.0.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.

- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI 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 BI distribution object (for advanced use cases) when `create_obj=True`.

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.logistic(loc=0.0, scale=1.0, sample=True)
```

Wrapper of:

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

Reference:

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

Low Rank Multivariate Normal

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.

Parameters: - *loc* (jnp.ndarray): Mean vector.

cov_factor (jnp.ndarray): Matrix used to construct the covariance matrix.

cov_diag (jnp.ndarray): Diagonal elements of the covariance matrix.

- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *shape* (tuple): 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 used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Example Usage:

from BI import bi m = bi('cpu') event_size = 100 # Our distribution has 100 dimensions rank = 5
m.dist.low_rank_multivariate_normal(- *loc*=m.dist.normal(0,1, shape = (event_size,), sample=True)*2, cov_factor=m.dist.normal(0,1, shape = (event_size, rank), sample=True), cov_diag=jnp.exp(m.dist.normal(0,1, shape = (event_size,), sample=True))* 0.1, sample=True)

Wrapper of:

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

Reference:

- Multivariate normal distribution – Wikipedia
- TensorFlow Probability: LowRankMultivariateNormal

```
bi.dist.low_rank_multivariate_normal(
loc,
cov_factor,
cov_diag,
validate_args=None,
```

```
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

Lower Truncated Power Law

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

Args:

```
bi.dist.lower_truncated_power_law(
alpha,
low,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- `alpha` (jnp.ndarray): index of the power law distribution. Must be less than -1.
- `low` (jnp.ndarray): lower bound of the distribution. Must be greater than 0.

- `shape` (tuple): 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.
- `event` (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- `mask` (jnp.ndarray, bool): Optional boolean array to mask observations.
- `create_obj` (bool): If True, returns the raw BI distribution object instead of creating a sample site.This is essential for building complex distributions like `MixtureSameFamily`.
- `sample` (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- `seed` (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- `obs` (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- `name` (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.lower_truncated_power_law(alpha=-2.0, low=1.0, sample=True)
```

Wrapper of:

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

Reference:

- Power-law distribution – Wikipedia
- Pareto distribution – Wikipedia
- Truncated Pareto distribution

Matrix Normal

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 ×p), with mean matrix

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

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

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

Also the equivalence to vec-form is given:

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

Args:

```
bi.dist.matrix_normal(
loc,
scale_tril_row,
scale_tril_column,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
```

```
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- `loc` (array_like): Location of the distribution.
- `scale_tril_row` (array_like): Lower cholesky of rows correlation matrix.
- `scale_tril_column` (array_like): Lower cholesky of columns correlation matrix.
- `shape` (tuple): 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.
- `event` (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- `mask` (jnp.ndarray, bool): Optional boolean array to mask observations.
- `create_obj` (bool): If True, returns the raw BI distribution object instead of creating a sample site.This is essential for building complex distributions like `MixtureSameFamily`.
- `sample` (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- `seed` (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- `obs` (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- `name` (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
import jax.numpy as jnp
m = bi('cpu')
n_rows, n_cols = 3, 4

- *loc* = jnp.zeros((n_rows, n_cols))
U_row_cov = jnp.array([[1.0, 0.5, 0.2],
[0.5, 1.0, 0.3],
[0.2, 0.3, 1.0]])
scale_tril_row = jnp.linalg.cholesky(U_row_cov)

V_col_cov = jnp.array([[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]])

# The argument passed to the distribution is its Cholesky factor
scale_tril_column = jnp.linalg.cholesky(V_col_cov)

m.dist.matrix_normal(
oc=loc,
scale_tril_row=scale_tril_row,
scale_tril_column=scale_tril_column,
sample=True
)
```

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

Reference:

- – https://en.wikipedia.org/wiki/Matrix_normal_distribution “Matrix normal distribution”
- – https://statproofbook.github.io/P/matn-pdf.html “Probability density function of the matrix-normal distribution | The Book of Statistical Proofs”

A marginalized finite mixture of component distributions.

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:

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

Args:

```
bi.dist.mixture(
mixing_distribution,
component_distributions,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=0,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *mixing_distribution* (BI.distribution): The distribution used to determine the mixing weights.
- *component_distributions* (list[BI.distribution]): The list of component distributions.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from jax import random
import BI as pyro
m = pyro.distributions.Mixture(
pyro.distributions.Categorical(torch.ones(2)),
[pyro.distributions.Normal(0, 1), pyro.distributions.Normal(2, 1)]
)
samples = m.sample(sample_shape=(10,))
```

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

References:

- https://en.wikipedia.org/wiki/Mixture_distribution "Mixture distribution"
- https://pymc3-testing.readthedocs.io/en/rtd-docs/notebooks/marginalized_gaussian_mixture_model.ht "Marginalized Gaussian Mixture Model — PyMC3 3.1rc3 documentation"

Mixture General

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 0) 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 BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.

- `shape` (tuple): 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 used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- **obs (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- **name (str, optional)**: The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- **When `sample=False`**: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

from BI import bi m = bi('cpu') m.dist.mixture_general(mixing_distribution=m.dist.categorical(probs=jnp.arr
0.7]), create_obj = True),
component_distributions=[m.dist.normal(loc=0.0, scale=1.0, create_obj=True),m.dist.normal
(loc=0.0, scale=1.0, create_obj=True)], sample = True)

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

Reference:

- – https://preliz.readthedocs.io/en/latest/distributions/gallery/mixture.html "Mixture Distribution — PreliZ 0.22.0 documentation"
- – https://reliability.readthedocs.io/en/latest/Mixture%20models.html "Mixture models — reliability 0.9.0 documentation"
- – https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.Mixture.html "Mixture — SciPy v1.16.2 Manual"

```
bi.dist.mixture_general(
mixing_distribution,
component_distributions,
support=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

Finite mixture of component distributions from the same family.

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 0), (_{i=1}^K w_i = 1). Let the component family have density (f(x _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.

Args:

```
bi.dist.mixture_same_family(
mixing_distribution,
component_distribution,
validate_args=None,
name='x',
obs=None,
mask=None,
```

```
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- **Distribution Args**:
- **mixing_distribution**: A `Categorical` distribution representing the mixing weights.
- **component_distributions**: A list of distributions representing the components of the mixture.
- **Sampling / Modeling Args**:
- `shape` (tuple): 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.
- `event` (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- `mask` (jnp.ndarray, bool): Optional boolean array to mask observations.
- `create_obj` (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- **sample (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- **seed (int, optional)**: An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- **obs (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- **name (str, optional)**: The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- **When `sample=False`**: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.mixture_same_family(
mixing_distribution=m.dist.categorical(probs=jnp.array([0.3, 0.7]), create_obj = True),
component_distribution=m.dist.normal(loc=0.0, scale=1.0, shape = (2,), create_obj=True),
sample = True
)
```

Wrapper of: https://num.pyro.ai/en/stable/distributions.html#mixture-, to_jax = True)same-family

Reference:

- Mixture distributions

Multinomial

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,…,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 $ X = (X_1, …, X_k)$ follows a multinomial distribution.

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

PMF (Probability Mass Function):

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

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

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

Args:

```
bi.dist.multinomial(
total_count=1,
probs=None,
logits=None,
total_count_max=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *total_count* (int or jnp.ndarray): The number of trials.
- *probs* (jnp.ndarray, optional): Event probabilities. Must sum to 1.
- *logits* (jnp.ndarray, optional): 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.
- *shape* (tuple): 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.

- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.multinomial(total_count=10, probs=jnp.array([0.2, 0.3, 0.5]), sample=True)
```

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

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"

Multinomial Logits

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 $\ell = (\ell_1, ..., \ell_K)$, 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.

Args:

```
bi.dist.multinomial_logits(
logits,
total_count=1,
total_count_max=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *logits* (jnp.ndarray): Logits for each outcome. Must be at least one-dimensional.
- *total_count* (jnp.ndarray): The total number of trials.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.multinomial_logits(logits=jnp.array([1.0, 0.5], dtype=jnp.float32), total_count=jnp.ar
```

Wrapper of:

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

Reference:

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

Multinomial Probs

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.

Args:

```
bi.dist.multinomial_probs(
probs,
total_count=1,
total_count_max=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *probs* (jnp.ndarray): Vector of probabilities for each outcome. Must sum to 1.
- *total_count* (jnp.ndarray): The number of trials.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.multinomial_probs(probs=jnp.array([0.2, 0.3, 0.5]), total_count=10, sample=True)
```

Wrapper of:

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

---

Multivariate Normal

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.

Args:

```
bi.dist.multivariate_normal(
loc=0.0,
covariance_matrix=None,
precision_matrix=None,
scale_tril=None,
validate_args=None,
name='x',
```

```
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* (tuple): The mean vector of the distribution.
- *covariance_matrix* (jnp.ndarray, optional): The covariance matrix of the distribution. Must be positive definite.
- *precision_matrix* (jnp.ndarray, optional): The precision matrix (inverse of the covariance matrix) of the distribution. Must be positive definite.
- *scale_tril* (jnp.ndarray, optional): The lower triangular Cholesky decomposition of the covariance matrix.
- `shape` (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.multivariate_normal(
- *loc*=jnp.array([1.0, 0.0, -2.0]),
covariance_matrix=jnp.array([[ 2.0,  0.7, -0.3],
[ 0.7,  1.0,  0.5],
[-0.3,  0.5,  1.5]]),
sample=True
)
```

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

Reference:

- Multivariate normal distribution – Wikipedia

Multivariate Student's t

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 ({x} R^d) is given by:

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

where:

- $ ( ) $ is the Gamma function.
- $ $ is the mean vector.
- $ $ is the scale (covariance) matrix.
- $ $ is the degrees of freedom.
- $ d $ is the dimensionality of $ {x} $.

Args:

```
bi.dist.multivariate_student_t(
df,
loc=0.0,
scale_tril=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *df* (jnp.ndarray): Degrees of freedom, must be positive.
- *loc* (jnp.ndarray): Location vector, representing the mean of the distribution. scale_tril (jnp.ndarray): Lower triangular matrix defining the scale.
- `shape` (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.

- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
import jax.numpy as jnp
m = bi('cpu')
m.dist.multivariate_student_t(
df = 2,
- *loc*=jnp.array([1.0, 0.0, -2.0]),
scale_tril=jnp.linalg.cholesky(
jnp.array([[ 2.0,  0.7, -0.3],
[ 0.7,  1.0,  0.5],
[-0.3,  0.5,  1.5]])),
sample=True
)
```

Wrapper of:

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

Reference:

- Multivariate Student-t distribution – Wikipedia

Negative Binomial

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,…)), with (r) successes fixed, success probability (p), failure probability (q = 1 - p):

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

Args:

```
bi.dist.negative_binomial(
total_count,
probs=None,
logits=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *total_count* (jnp.ndarray): The total number of events.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.

- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.negative_binomial(total_count=5.0,probs = jnp.array([0.2, 0.3, 0.5]), sample=True)
```

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

References:

- https://en.wikipedia.org/wiki/Negative_binomial_distribution
- https://www.statology.org/negative-binomial-distribution/

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 (= {E}[Y]) and (>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:

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

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

Args:

```
bi.dist.negative_binomial2(
mean,
concentration,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
```

```
to_jax=True,
)
```

- *mean* (jnp.ndarray or float): The mean of the distribution. This is equivalent to the `mu` parameter.
- *concentration* (jnp.ndarray or float): The concentration parameter. This is equivalent to the *alpha* parameter.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI NegativeBinomial distribution object: When `sample=False` (for model building). jnp.ndarray: A JAX array of samples drawn from the NegativeBinomial distribution (for direct sampling). BI NegativeBinomial distribution object: When `create_obj=True` (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.negative_binomial2(mean=2.0, concentration=3.0, sample=True)
```

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

Reference:

- [Negative Binomial Regression]9https://www.cambridge.org/core/books/negative-binomial-regression/12D6281A46B9A980DC6021080C9419E7)
- Negative Binomial Regression: Second Edition

Negative Binomial Logits

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.

Args:

```
bi.dist.negative_binomial_logits(
total_count,
logits,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *total_count* (jnp.ndarray): The parameter controlling the shape of the distribution. Represents the total number of trials.
- *logits* (jnp.ndarray): The log-odds parameter. Related to the probability of success.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

Negative Binomial Logits: A BI Negative Binomial Logits distribution object (for model building).

jnp.ndarray: A JAX array of samples drawn from the Negative Binomial Logits distribution (for direct sampling).

Negative Binomial Logits: The raw BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.negative_binomial_logits(total_count=5.0, logits=0.0, sample=True)
```

Wrapper of:

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

---

Negative Binomial with probabilities.

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'.

Args:

```
bi.dist.negative_binomial_probs(
total_count,
probs,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *total_count* (jnp.ndarray): A numeric vector, matrix, or array representing the parameter.
- *probs* (jnp.ndarray): A numeric vector representing event probabilities. Must sum to 1.

- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.negative_binomial_probs(total_count=10.0, probs = jnp.array([0.2, 0.3, 0.5]), sample=
```

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

---

Normal

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.

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

Args:

```
bi.dist.normal(
loc=0.0,
scale=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* (jnp.ndarray): The mean of the distribution.
- *scale* (jnp.ndarray): The standard deviation of the distribution.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.

- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.normal(loc=0.0, scale=1.0, sample=True)
```

Wrapper of:

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

References:

- Wikipedia: Normal distribution – PDF
- Wolfram MathWorld: Normal Distribution

Ordered Logistic

The ordered logistic distribution is used for modeling **ordinal** outcome variables $ Y 1,2,…,K $ (i.e., categories with a natural order) via a latent continuous predictor $ $ and a set of increasing *cut-points*. When $ $ crosses thresholds, the observed Y

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

Args:

```
bi.dist.ordered_logistic(
predictor,
cutpoints,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *predictor* (jnp.ndarray): Prediction in real domain; typically this is output of a linear model.
- *cutpoints* (jnp.ndarray): Positions in real domain to separate categories.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.

- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.ordered_logistic(predictor=jnp.array([0.2, 0.5, 0.8]), cutpoints=jnp.array([-1.0, 0.0
```

Wrapper of:

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

Reference:

- https://en.wikipedia.org/wiki/Ordered_logit
- https://mc-stan.org/docs/2_21/functions-reference/ordered-logistic-distribution.html
- https://labdisia.disia.unifi.it/grilli/files/Papers/Ordered_Logit.pdf"

Pareto

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** $ > 0 $ (which controls the tail heaviness).

A random variable $ X Pareto(, x_m) $ takes values $ x x_m $.

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

Args:

```
bi.dist.pareto(
scale,
alpha,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *scale* (jnp.ndarray or float): Scale parameter of the Pareto distribution. Must be positive.
- *alpha* (jnp.ndarray or float): Shape parameter of the Pareto distribution. Must be positive.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).

- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.pareto(scale=2.0, alpha=3.0, sample=True)
```

Wrapper of:

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

Reference:

- Wikipedia: Pareto distribution – PDF
- Arnold, B. C. (2015). *Pareto Distributions*, Second Edition. CRC Press.

Poisson

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 $ $ > 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:

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

Args:

```
bi.dist.poisson(
rate,
is_sparse=False,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *rate* (jnp.ndarray): The rate parameter, representing the average number of events.

- *is_sparse* (bool, optional): 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`.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.poisson(rate=2.0, sample=True)
```

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

References:

- Wikipedia: Poisson distribution – PMF
- Johnson, N. L., Kotz, S., & Kemp, A. W. (1992). *Univariate Discrete Distributions* (2nd ed.). Wiley.

Projected Normal

The projected normal distribution arises by taking a multivariate normal vector $ X N (,) $ in $ 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.

Args:

```
bi.dist.projected_normal(
concentration,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *concentration* (jnp.ndarray): The concentration parameter, representing the direction towards which the samples are concentrated. Must be a JAX array with at least one dimension.
- *shape* (tuple): 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.

- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.projected_normal(concentration=jnp.array([1.0, 3.0, 2.0]), sample=True)
```

Wrapper of:

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

---

Relaxed Bernoulli

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:

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

Args:

```
bi.dist.relaxed_bernoulli(
temperature,
probs=None,
logits=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *temperature* (jnp.ndarray): The temperature parameter. Must be greater than 0.

- *probs* (jnp.ndarray, optional): The probability of success. Must be in the interval `[0, 1]`. Only one of `probs` or `logits` can be specified.
- *logits* (jnp.ndarray, optional): The log-odds of success. Must be in the interval `[-inf, inf]`. Only one of `probs` or `logits` can be specified.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI 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 BI distribution object (for advanced use cases) when `create_obj=True`.

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.relaxed_bernoulli(temperature=1.0, probs = jnp.array([0.2, 0.3, 0.5]), sample=True)
```

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

References

- https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/RelaxedBernoulli "tfp.distributions.RelaxedBernoulli | TensorFlow Probability"
- https://rstudio.github.io/tfprobability/reference/tfd_relaxed_bernoulli.html "Relaxed-Bernoulli distribution with temperature and logits …"

Relaxed Bernoulli Logits

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 $\to 0$ the distribution approximates a Bernoulli; as temperature $\to \infty$ 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 = ( ), and temperature = (> 0), then one generates

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

where $ ( ) $ 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).

Args:

```
bi.dist.relaxed_bernoulli_logits(
temperature,
logits,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *temperature* (jnp.ndarray): The temperature parameter, must be positive.
- *logits* (jnp.ndarray): The logits parameter.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

RelaxedBernoulliLogits: A BI RelaxedBernoulliLogits distribution object (for model building). jnp.ndarray: A JAX array of samples drawn from the RelaxedBernoulliLogits distribution (for direct sampling). RelaxedBernoulliLogits: The raw BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.relaxed_bernoulli_logits(temperature=1.0, logits=0.0, sample=True)
```

Wrapper of:

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

Reference:

- https://arxiv.org/pdf/1611.00712

Right Truncated

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:

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

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:

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

Args:

```
bi.dist.right_truncated_distribution(
base_dist,
high=0.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *base_dist*: The base distribution to truncate. Must be a univariate distribution with real support.
- *high* (float, jnp.ndarray, optional): The upper truncation point. The support of the new distribution is $-\infty$, high. Defaults to 0.0.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.

- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.right_truncated_distribution(base_dist = m.dist.normal(0,1, create_obj = True), high=
```

Wrapper of:

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

References:

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

---

Sine Bivariate Von Mises

In probability theory and statistics, the bivariate von Mises distribution is a probability distribution describing values on a torus. It may be thought of as an analogue on the torus of the bivariate normal distribution. The distribution belongs to the field of directional statistics. The general bivariate von Mises distribution was first proposed by Kanti Mardia in 1975. One of its variants is today used in the field of bioinformatics to formulate a probabilistic model of protein structure in atomic detail, such as backbone-dependent rotamer libraries.

Args:

```
bi.dist.sine_bivariate_vonmises(
phi_loc,
psi_loc,
phi_concentration,
psi_concentration,
correlation=None,
weighted_correlation=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
)
```

- *phi_loc* (jnp.ndarray): The location parameter for the first angle (phi).
- *psi_loc* (jnp.ndarray): The location parameter for the second angle (psi).
- *phi_concentration* (jnp.ndarray): The concentration parameter for the first angle (phi). Must be positive.
- *psi_concentration* (jnp.ndarray): The concentration parameter for the second angle (psi). Must be positive.
- *correlation* (jnp.ndarray, optional): The correlation parameter between the two angles. One of `correlation` or `weighted_correlation` must be specified.
- *weighted_correlation* (jnp.ndarray, optional): An alternative correlation parameter. One of `correlation` or `weighted_correlation` must be specified.

- *validate_args* (bool, optional): Whether to enable validation of distribution parameters. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations. If provided, events with a `True` mask will be conditioned on `obs`, while the remaining events will be treated as latent variables. Defaults to `None`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *shape* (tuple, optional): A multi-purpose argument for shaping. When `sample=False` (model building), this is used with `.expand(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.
- *event* (int, optional): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to `False`.

Returns:

BI.primitives.Messenger: A BI sample site object when used in a model context (`sample=False`). jnp.ndarray: A JAX array of samples drawn from the SineBivariateVonMises distribution (for direct sampling, `sample=True`). numpyro.distributions.Distribution: The raw BI distribution object (if `create_obj=True`).

Example Usage:

```
from BI import bi
import jax.numpy as jnp
m = bi('cpu')

# Direct sampling
samples = m.dist.sine_bivariate_vonmises(
phi_loc=0.0,
psi_loc=jnp.pi,
phi_concentration=1.0,
psi_concentration=1.0,
correlation=0.5,
sample=True,
- *shape*=(10,)
)

# Usage within a model
def my_model():
angles = m.dist.sine_bivariate_vonmises(
phi_loc=0.0,
psi_loc=0.0,
phi_concentration=2.0,
psi_concentration=2.0,
weighted_correlation=0.9,
name='angles'
)
# ... rest of the model
```

Wrapper of:

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

---

Sine-skewing

The sine-skewed von Mises distribution is an extension of the symmetric circular (or toroidal) von Mises (or bivariate von Mises) distribution to allow for skewness (asymmetry) via a sine-based skewing function. It is used to model directional data on the circle (or torus) that depart from symmetry.1

Parameters:

- **base_dist:** Base density on a d-dimensional torus. Supported base distributions include: 1D :class:`~numpyro.distributions.VonMises`, :class:`~numnumpyro.distributions.SineBivariateVonM 1D :class:`~numpyro.distributions.ProjectedNormal`, and :class:`~numpyro.distributions.Uniform` (-pi, pi).
- **skewness:** Skewness of the distribution.
- **sample (bool, optional):** A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- **seed (int, optional):** An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- **obs (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- **name (str, optional):** The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

PDF:

The probability density function (PDF) of the Sine Skewed X distribution is not explicitly defined here, but it is derived from the base distribution and the skewness parameter.

Example Usage:

from num.pyro import distributions as dist import num.pyro as pyro import num.numpy as np

m = pyro.distributions.Normal(loc=0.0, scale=1.0) skewness = np.array([0.5, 0.5]) sine_skewed = dist.SineSkewed(base_dist=m, skewness=skewness) samples = sine_skewed.sample((1000,))

```
bi.dist.sine_skewed(
base_dist: numpyro.distributions.distribution.Distribution,
skewness,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
```

```
create_obj=False,
to_jax=True,
)
```

SoftLaplace

Smooth distribution with Laplace-like tail behavior.

This distribution corresponds to the log-convex density:

$$z = (value - loc)/scalelog_{p}rob = log(2/pi) - log(scale) - logaddexp(z, -z)$$

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.

Args:

```
bi.dist.soft_laplace(
loc,
scale,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc*: Location parameter.
- *scale*: Scale parameter.

- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.soft_laplace(loc=0.0, scale=1.0, sample=True)
```

Wrapper of:

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

Student's t

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 $ $, $it converges to the Normal distribution.$ X t_ (,) $$ where:

- $ $ is the **location (mean)** parameter
- $ > 0 $ is the **scale** parameter
- $ > 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 $ \to\infty$, this approaches $ {N}(, ^2) $.

Args:

```
bi.dist.student_t(
df,
loc=0.0,
scale=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *df* (jnp.ndarray): Degrees of freedom, must be positive.
- *loc* (jnp.ndarray): Location parameter, defaults to 0.0.
- *scale* (jnp.ndarray): Scale parameter, defaults to 1.0.
- *shape* (tuple): A multi-purpose argument for shaping. When `sample=False` (model building), this is used with `.expand(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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.student_t(df = 2, loc=0.0, scale=1.0, sample=True)
```

Wrapper of:

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

References:

- Wikipedia – Student's t distribution: PDF

Truncated Cauchy

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:

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

and $f_T(x) = 0$ for $x < a$ or $x > b$.

Here F_{Cauchy} is the standard Cauchy CDF $with parameters μ, γ).

Args:

```
bi.dist.truncated_cauchy(
loc=0.0,
scale=1.0,
low=None,
high=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
```

```
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* (float): Location parameter of the Cauchy distribution.
- *scale* (float): Scale parameter of the Cauchy distribution.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI 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 BI distribution object (for advanced use cases) when `create_obj=True`.

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.truncated_cauchy(loc=0.0, scale=1.0, sample=True)
```

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

References:

- https://en.wikipedia.org/wiki/Cauchy_distribution "Cauchy distribution"
- https://www.tandfonline.com/doi/full/10.1080/00207390600595223 "Full article: A truncated Cauchy distribution"
- https://rstudio.github.io/tfprobability/reference/tfd_truncated_cauchy.html "The Truncated Cauchy distribution. — tfd_truncated_cauchy"

Truncated

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 $(Y = X \mid a < X \leq b)$ 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). * If both bounds are finite (i.e., doubly truncated), that is a valid and often-used scenario.

Args:

```
bi.dist.truncated_distribution(
base_dist,
low=None,
high=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *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.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.

- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.truncated_distribution(base_dist = m.dist.normal(0,1, create_obj = True), high=1, low
```

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

References:

- https://en.wikipedia.org/wiki/Truncated_distribution "Truncated distribution"
- https://pages.stern.nyu.edu/~wgreene/Text/Edition8/PDF/M19_GREE1366_08_SE_C19.pdf "19"
- https://search.r-project.org/CRAN/refmans/LaplacesDemon/html/dist.Truncated.html "R: Truncated Distributions"
- https://dspace.cuni.cz/bitstream/handle/20.500.11956/127921/130308341.pdf?isAllowed=y&sequence=1 "Truncated data thesis"

Truncated Normal

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

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

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 normalises the density so the total probability over $([a, b])$ is 1.

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

Args:

```
bi.dist.truncated_normal(
loc=0.0,
scale=1.0,
low=None,
high=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *loc* (float): The location parameter of the normal distribution.

- *sample* (float): The scale parameter of the normal distribution.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.truncated_normal(loc=0.0, scale=1.0, sample=True)
```

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

References:

- https://en.wikipedia.org/wiki/Truncated_normal_distribution "Truncated normal distribution"
- https://people.math.sc.edu/burkardt/m_src/truncated_normal/truncated_normal.html "TRUNCATED_NORMAL - The Truncated Normal Distribution"
- https://www.statisticshowto.com/truncated-normal-distribution/ "Truncated Distribution / Truncated Normal Distribution - Statistics ..."
- https://real-statistics.com/normal-distribution/truncated-normal-distribution/ "Truncated Normal Distribution | Real Statistics Using Excel"
- https://stats.stackexchange.com/questions/525894/understanding-the-pdf-of-a-truncated-normal-distribution "Understanding the pdf of a truncated normal distribution"

Truncated PolyaGamma

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.

Args:

```
bi.dist.truncated_polya_gamma(
batch_shape=(),
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *batch_shape* (tuple): The shape of the batch dimension.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions.
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.

- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.truncated_polya_gamma(batch_shape=(), sample=True)
```

Wrapper of:

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

---

Two Sided Truncated

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.

Args:

```
bi.dist.two_sided_truncated_distribution(
base_dist,
low=0.0,
high=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *base_dist*: The base distribution to truncate.
- *low*: The lower bound for truncation.
- *high*: The upper bound for truncation.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.

- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

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

---

Uniform

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.

$$f(x) = \begin{cases} \frac{1}{b-a}, a \leq x \leq b, [2mm]0, \text{otherwise.} \end{cases}$$

Args:

```
bi.dist.uniform(
low=0.0,
high=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
```

```
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *low* (jnp.ndarray): The lower bound of the uniform interval.
- *high* (jnp.ndarray): The upper bound of the uniform interval.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to False.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI 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 BI distribution object (for advanced use cases) when `create_obj=True`.

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.uniform(low=0.0, high=1.0, sample=True)
```

Wrapper of:

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

References:

- Wikipedia – Uniform distribution (continuous)

Unit

- The Unit distribution is the simplest continuous probability distribution.
- Every number in ([0,1]) is equally likely.
- Often used in simulation, Monte Carlo methods, and as a building block for generating other distributions via the inverse-CDF method.

$$f(x) = \Big\{1, 0 \leq x \leq 1, [1mm]0, \text{otherwise.}$$

Args:

```
bi.dist.unit(
log_factor,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *log_factor* (jnp.ndarray): Log factor for the unit distribution. This parameter determines the *shape* and batch size of the distribution.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI Unit distribution object: When `sample=False` (for model building). jnp.ndarray: A JAX array of samples drawn from the Unit distribution (for direct sampling). BI Unit distribution object: When `create_obj=True` (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.unit(log_factor=jnp.ones(5), sample=True)
```

Wrapper of:

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

Weibull

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)

$$f(x \mid \lambda, k) = \left\{ \frac{k}{\lambda} \left(\frac{x}{\lambda} \right)^{k-1} e^{-(x/\lambda)^k}, x \geq 0, [2mm]0, x < 0 \right.$$

Args:

```
bi.dist.weibull(
scale,
concentration,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *scale* (jnp.ndarray): The scale parameter of the Weibull distribution. Must be positive.
- *concentration* (jnp.ndarray): The concentration parameter of the Weibull distribution. Must be positive.

- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI 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 BI distribution object (for advanced use cases) when `create_obj=True`.

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.weibull(scale=1.0, concentration=2.0, sample=True)
```

Wrapper of:

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

References:

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

Wishart

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.

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

where:

- p = dimension of the square matrices
- $V \in R^{p \times p}$ = positive definite **scale matrix** (`scale_matrix`)
- $n \geq p$ = **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.

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

where:

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

Args:

```
bi.dist.wishart(
concentration,
scale_matrix=None,
rate_matrix=None,
scale_tril=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *concentration* (jnp.ndarray): 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* (jnp.ndarray, optional): Scale matrix analogous to the inverse rate of a Gamma distribution.
- *rate_matrix* (jnp.ndarray, optional): Rate matrix anaologous to the rate of a Gamma distribution.
- *scale_tril* (jnp.ndarray, optional): Cholesky decomposition of the :code:`scale_matrix`.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations.
- *create_obj* (bool, optional): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.

- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.wishart(concentration=5.0, scale_matrix=jnp.eye(2), sample=True)
```

Wrapper of:

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

References:

- Wikipedia – Wishart distribution

Wishart Cholesky

- 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 ( ).

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).

Parameters

- concentration: (Tensor) Positive concentration parameter analogous to the concentration of a :class:`Gamma` distribution. The concentration must be larger than the dimensionality of the scale matrix.
- scale_matrix: (Tensor, optional) Scale matrix analogous to the inverse rate of a :class:`Gamma` distribution. If not provided, `rate_matrix` or `scale_tril` must be.
- rate_matrix: (Tensor, optional) Rate matrix anaologous to the rate of a :class:`Gamma` distribution. If not provided, `scale_matrix` or `scale_tril` must be.

- scale_tril: (Tensor, optional) Cholesky decomposition of the :code:`scale_matrix`. If not provided, `scale_matrix` or `rate_matrix` must be.
- sample (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- seed (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- obs (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- name (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

References:

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

```
bi.dist.wishart_cholesky(
concentration,
scale_matrix=None,
rate_matrix=None,
scale_tril=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

Generic Zero Inflated

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):

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

Args:

```
bi.dist.zero_inflated_distribution(
base_dist,
gate=None,
gate_logits=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
```

```
create_obj=False,
to_jax=True,
)
```

- *base_dist* (Distribution): The base distribution to be zero-inflated (e.g., Poisson, NegativeBinomial).
- *gate* (jnp.ndarray, optional): Probability of extra zeros (between 0 and 1).
- *gate_logits* (jnp.ndarray, optional): Log-odds of extra zeros.
- *validate_args* (bool, optional): Whether to validate parameter values. Defaults to None.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.zero_inflated_distribution(base_dist=m.dist.poisson(rate=5, create_obj = True), gate =
```

Wrapper of: https://num.pyro.ai/en/stable/distributions.html#zeroinflateddistribution

References:

- https://en.wikipedia.org/wiki/Zero-inflated_model "Zero-inflated model"
- https://larmarange.github.io/guide-R/analyses_avancees/modeles-zero-inflated.html "48 Modèles de comptage zero-inflated et hurdle – guide-R"

Zero-Inflated Negative Binomial

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 $ $ (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 $ $ and dispersion parameter $ $ 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,…} $. Denote:

- π = probability of being in the "always zero" (structural zero) process, with $ 0 1$.
- The count process is NB with parameters: mean $\mu > 0$ and dispersion/shape parameter $\alpha > 0$ (or equivalently size parameter (r)). Then:

$$P(X = 0) = \pi; +; (1 - \pi); P_{\text{NB}}(0; \mu, \alpha),$$
$$P(X = k) = (1 - \pi); P_{\text{NB}}(k; \mu, \alpha), \quad k = 1, 2, 3, ...$$

Here $P_{\text{NB}}(k; \mu, \alpha)$ is the PMF of the negative binomial distribution for count k.

Args:

```
bi.dist.zero_inflated_negative_binomial2(
mean,
concentration,
gate=None,
gate_logits=None,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *mean* (jnp.ndarray or float): The mean of the Negative Binomial 2 distribution.
- *concentration* (jnp.ndarray or float): The concentration parameter of the Negative Binomial 2 distribution.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.

- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.zero_inflated_negative_binomial2(mean=2.0, concentration=3.0, gate = 0.3, sample=True)
```

Wrapper of: https://num.pyro.ai/en/stable/distributions.html#zeroinflatednegativebinomial2

References:

- https://www.ewadirect.com/proceedings/tns/article/view/27624 "Social Networks Count Data: Negative Binomial Distributions and Extensions Versus Poisson and Bernoulli Models"
- https://pmc.ncbi.nlm.nih.gov/articles/PMC7880198/ "Bayesian Zero-Inflated Negative Binomial Regression Based on Pólya-Gamma Mixtures - PMC"
- https://search.r-project.org/CRAN/refmans/emdbook/html/dzinbinom.html "R: Zero-inflated negative binomial distribution"

A Zero Inflated Poisson

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:

$$P(X = k) = \begin{cases} \pi; +; (1 - \pi), e^{-\lambda}, \text{if } k = 0, [6pt](1 - \pi), \dfrac{\lambda^k e^{-\lambda}}{k!}, \text{if } k = 1, 2, 3, ... \end{cases}$$

(Here the Poisson component contributes also a mass at zero of $(1 - \pi)e^{-\lambda}$, plus the extra π mass.)

Args:

```
bi.dist.zero_inflated_poisson(
gate,
rate=1.0,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
```

```
to_jax=True,
)
```

- *gate* (jnp.ndarray): The probability of observing a zero.
- *rate* (jnp.ndarray): The rate parameter of the underlying Poisson distribution.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BI ZeroInflatedPoisson distribution object (when `sample=False`). JAX array of samples drawn from the ZeroInflatedPoisson distribution (when `sample=True`). The raw BI distribution object (when `create_obj=True`).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.zero_inflated_poisson(gate = 0.3, rate=2.0, sample=True)
```

Wrapper of: https://num.pyro.ai/en/stable/distributions.html#zeroinflatedpoisson

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"

Zero Sum Normal

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 = _i x_i $).

Args:

```
bi.dist.zero_sum_normal(
scale,
event_shape,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=None,
shape=(),
event=0,
create_obj=False,
to_jax=True,
)
```

- *scale* (array_like): Standard deviation of the underlying normal distribution before the zerosum constraint is enforced.

- *event_shape* (tuple): Shape of the event dimension of the distribution.
- *shape* (tuple): 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.
- *event* (int): The number of batch dimensions to reinterpret as event dimensions (used in model building).
- *mask* (jnp.ndarray, bool): Optional boolean array to mask observations.
- *create_obj* (bool): If True, returns the raw BI distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`.
- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the BI model context. If `False`, it will create a `BI.sample` site within a model. Defaults to `False`.
- *seed* (int, optional): An integer used to generate a JAX PRNGKey for reproducible sampling when `sample=True`. This argument has no effect when `sample=False`, as randomness is handled by BI's inference engine. Defaults to 0.
- *obs* (jnp.ndarray, 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 `None`, the site is treated as a latent (unobserved) random variable. Defaults to `None`.
- *name* (str, optional): The name of the sample site in a BI model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A BI 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 BI distribution object (for advanced use cases).

Example Usage:

```
from BI import bi
m = bi('cpu')
m.dist.zero_sum_normal(scale=1.0, event_shape = (2,), sample = True)
```

Wrapper of:

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

References:

https://www.pymc.io/projects/docs/en/latest/api/distributions/generated/pymc.ZeroSumNormal.html

---