Sampled from distributions

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

## Methods

### LowerTruncatedPowerLaw

Lower truncated power law distribution with :math:`\alpha` index.

The probability density function (PDF) is given by:

```
\begin{align*}
f(x; \alpha, a) = (-\alpha-1)a^{-\alpha - 1}x^{-\alpha},
\qquad x \geq a, \qquad \alpha < -1,
\end{align*}
```

where :math:`a` is the lower bound.

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

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

## Methods

### AsymmetricLaplace

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).

```
\begin{align*}
f(x) = \frac{\text{asymmetry}}{\text{scale}(\text{asymmetry}^2+1)} \exp\left(-\frac{\text{as
\text{if } x < \text{loc} \\
\frac{\text{asymmetry}}{\text{scale}(\text{asymmetry}^2+1)} \exp\left(-\frac{1}{\text{scale}
\text{if } x \ge \text{loc}
\end{align*}
```

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

- *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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro 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 NumPyro 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 NumPyro distribution object (for advanced use cases).

Example Usage:

~~~
```python
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
# Sampled from distributions
 `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a

 ## Methods

 ### AsymmetricLaplaceQuantile

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.

```{latex}
\begin{align*}
f(x) = \frac{1}{2 \sigma} \exp\left(-\frac{|x - \mu|}{\sigma} \frac{1}{q-1}\right) \left(1 -
\end{align*}
~~~

Args: - *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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro 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 NumPyro 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

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

Sampled from distributions

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

## Methods

### Bernoulli distribution.

The Bernoulli distribution models a single trial with two possible outcomes: success or failure. It is parameterized by the probability of success, often denoted as ‘p’.

```
\begin{align*}
P(X=1) = p \\
P(X=0) = 1 - p
\end{align*}
```

Args:

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

- *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 `.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.
- *create_obj* (bool, optional): If True, returns the raw NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

NumPyro 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 NumPyro 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

Sampled from distributions

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

## Methods

### BernoulliLogits

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.

```
\begin{align*}
P(x) = \sigma(logits)
\end{align*}
```

Args:

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

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 `.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). 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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro 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 NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### BernoulliProbs

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.

```
\begin{align*}
P(X=1) = p
\end{align*}
```

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

- *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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

BernoulliProbs: A NumPyro BernoulliProbs distribution object (for model building).

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

NumPyro distribution object: The raw NumPyro distribution 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Beta Distribution

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, often referred to as concentration parameters in the NumPyro context.

```
\begin{align*}
f(x) = \frac{x^{\alpha - 1} (1 - x)^{\beta - 1}}{B(\alpha, \beta)}
\end{align*}
```

where :num:math:`\alpha` and :num:math:`\beta` are the concentration parameters, and :num:math:`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=0,
shape=(),
event=0,
create_obj=False,
)
```

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

Beta: A NumPyro Beta distribution object (for model building). jnp.ndarray: A JAX array of samples drawn from the Beta distribution (for direct sampling). Beta: The raw NumPyro distribution 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### BetaBinomial Distribution

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.

```
\begin{align*}
P(X=k) = \binom{n}{k} \frac{\Gamma(\alpha + k)}{\Gamma(\alpha + \beta + n - k)} \frac{\Gamma
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro 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 NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### BetaProportion distribution.

The BetaProportion 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.

```
\begin{align*}
f(x) = \frac{x^{\alpha - 1} (1 - x)^{\beta - 1}}{B(\alpha, \beta)}
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

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

## Methods

### Binomial distribution.

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.

```
\begin{align*}
P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}
\end{align*}
```

Args:

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

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 `.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.
- *create_obj* (bool, optional): If True, returns the raw NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro 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 NumPyro distribution object (for advanced use cases) when `create_obj=True`.

Example Usage:

```python
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

Sampled from distributions

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

## Methods

### BinomialLogits Distribution

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.

```
\begin{align*}
P(X=k) = \binom{n}{k} \frac{e^{logits_k}}{1 + e^{logits_k}}
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro BinomialLogits distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the BinomialLogits distribution (for direct sampling).
- When `create_obj=True`: The raw NumPyro distribution 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#binomiallogits # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### BinomialProbs

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.

```
\begin{align*}
P(k) = \binom{n}{k} p^k (1-p)^{n-k}
\end{align*}
```

Args:

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

- *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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

NumPyro BinomialProbs distribution object (for model building).

JAX array of samples drawn from the BinomialProbs distribution (for direct sampling).

The raw NumPyro distribution 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Conditional Autoregressive (CAR) Distribution

The CAR distribution models a vector of variables where each variable is a linear combination of its neighbors in a graph.

```
\begin{align*}
p(x) = \prod_{i=1}^{K} \mathcal{N}(x_i | \mu_i, \Sigma_i)
\end{align*}
```

where :math:`\mu_i` is a function of the values of the neighbors of site :math:`i` and :math:`\Sigma_i` is the variance of site :math:`i`.

.. note::

The CAR distribution is a special case of the multivariate normal distribution. It is used to model spatial data, such as temperature or precipitation.

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

- *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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'. # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Categorical distribution.

The Categorical distribution, also known as the multinomial distribution, describes the probability of different outcomes from a finite set of possibilities. It is commonly used to model discrete choices or classifications.

```
\begin{align*}
P(k) = \frac{e^{\log(p_k)}}{\sum_{j=1}^{K} e^{\log(p_j)}}

where :math:`p_k` is the probability of outcome :math:`k`, and the sum is over all possible
\end{align*}
```

Args:

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

- *probs* (jnp.ndarray): A 1D array of probabilities for each category. Must sum to 1.

Distribution #### Args: None

Sampling / Modeling #### Args: - *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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro 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 NumPyro 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

Sampled from distributions

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

## Methods

### Categorical Logits Distribution

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.

```
\begin{align*}
P(k) = \frac{e^{logits_k}}{\sum_{j=1}^{K} e^{logits_j}}
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

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

## Methods

### 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.

```
\begin{align*}
P(x) = \frac{probs_i}{\sum_{k=1}^{K} probs_k}
\end{align*}
```

Args:

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

- *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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

NumPyro CategoricalProbs distribution object (for model building). JAX array of samples drawn from the CategoricalProbs distribution (for direct sampling). The raw NumPyro distribution 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Cauchy Distribution

Samples from a Cauchy distribution.

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.

```
\begin{align*}
f(x) = \frac{1}{\pi \gamma} \left[ \frac{\gamma^2}{(x - \mu)^2 + \gamma^2} \right]
\end{align*}
```

Args:

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

- *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 `.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). 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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to ‘x’.

Returns:

- When `sample=False`: A NumPyro 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 NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Chi-squared distribution.

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), which determines the shape of the distribution.

```
\begin{align*}
p(x; df) = \frac{1}{2^{df/2} \Gamma(df/2)} x^{df/2 - 1} e^{-x/2}
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

NumPyro Chi2 distribution object (when `sample=False`). JAX array of samples drawn from the Chi2 distribution (when `sample=True`). The raw NumPyro distribution object (when `create_obj=True`).

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

## Methods

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

```
\begin{align*}

p\left(\mathbf{x}\mid\boldsymbol{\mu},\mathbf{C}\right) =
\frac{\left(\mathrm{det}\,\mathbf{C}\right)^{-1/2}}{\left(2\pi\right)^{n / 2}}
\exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}\right)^\intercal
\mathbf{C}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}\right)\right),
\end{align*}
```

where :math:`\mathrm{det}` denotes the determinant and :math:`^\intercal` the transpose. Circulant matrices can be diagnolized efficiently using the discrete Fourier transform [1], allowing the log likelihood to be evaluated in :math:`n \log n` time for :math:`n` observations [2].

:param loc: Mean of the distribution :math:`\boldsymbol{\mu}`. :param covariance_row: First row of the circulant covariance matrix :math:`\boldsymbol{C}`. Because of periodic boundary conditions, the covariance matrix is fully determined by its first row (see :func:`jax.scipy.linalg.toeplitz` for further details). :param covariance_rfft: Real part of the real fast Fourier transform of :code:`covariance_row`, the first row of the circulant covariance matrix :math:`\boldsymbol{C}`.

- *sample* (bool, optional): A control-flow argument. If `True`, the function will directly sample a raw JAX array from the distribution, bypassing the NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro 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 :math:`\left[0, 1\right]^d`. *Journal of Computational and Graphical Statistics*, 3(4), 409–432. https://doi.org/10.1080/10618600.1994.10474655

Parameters

- *loc* : jnp.ndarray Mean of the distribution :math:`\boldsymbol{\mu}`.

covariance_row : jnp.ndarray, optional

First row of the circulant covariance matrix :math:`\mathbf{C}`. Defaults to None.

covariance_rfft : jnp.ndarray, optional Real part of the real fast Fourier transform of :code:`covariance_row`. Defaults to None.

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

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

Sampled from distributions

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

## Methods

### Delta distribution.

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.

```
\begin{align*}
P(x = v) = 1
\end{align*}
```

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

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

## Methods

### 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.

```
\begin{align*}
P(x_1, ..., x_K) = \frac{\Gamma(\sum_{i=1}^K \alpha_i)}{\prod_{i=1}^K \Gamma(\alpha_i)} \proc
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

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

Sampled from distributions

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

## Methods

### Dirichlet-Multinomial Distribution.

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 probability mass function is given by:

```
\begin{align*}
P(\mathbf{x} | \boldsymbol{\alpha}, n) = \frac{n!}{\prod_{i=1}^k x_i!} \frac{\Gamma(\sum_{i=
\end{align*}
```

where :math:`\mathbf{x}` is a vector of counts, :math:`n` is the total number of trials (`total_count`), and :math:`\boldsymbol{\alpha}` is the `concentration` parameter vector for the Dirichlet prior.

Args:

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

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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to `False`.

Returns:

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

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

## Methods

### Discrete Uniform Distribution

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.

```
\begin{align*}
P(X = k) = \frac{1}{high - low + 1}, \quad k \in \{low, low+1, ..., high\}
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro DiscreteUniform distribution object (for model building).
- When `sample=True`: A JAX array of samples drawn from the Discrete Uniform distribution (for direct sampling).
- When `create_obj=True`: The raw NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Doubly Truncated Power Law distribution.

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.

The probability density function (PDF) is defined as:

```
\begin{align*}
f(x; \\alpha, a, b) = \\frac{x^{\\alpha}}{Z(\\alpha, a, b)}
\end{align*}
```

where the normalization constant :math:`Z(\\alpha, a, b)` is given by:

```
\begin{align*}
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}
\end{align*}
```

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

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, optional): The shape of the output tensor. 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

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)
```

Sampled from distributions

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

## Methods

### Euler–Maruyama Distribution 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.

```
\begin{align*}
dX_t = f(X_t, t) dt + g(X_t, t) dW_t
\end{align*}
```

where: - :math:`X_t` is the state of the system at time :math:`t`. - :math:`f(X_t, t)` is the drift coefficient. - :math:`g(X_t, t)` is the diffusion coefficient. - :math:`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,
)
```

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, optional): The shape of the output tensor. Defaults to None.

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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro 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=
```

Sampled from distributions

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

## Methods

### Exponential distribution.

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.

```
\begin{align*}
f(x) = \lambda e^{-\lambda x} \text{ for } x \geq 0
\end{align*}
```

Args:

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

rate (jnp.ndarray): The rate parameter, :math:`\lambda`. Must be positive.

- *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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro 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 NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Folded distribution

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.

```
\begin{align*}
p(x) = \sum_{k=-\infty}^{\infty} p(x - 2k)
\end{align*}
```

Args:

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

- *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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

## Methods

### Gamma Distribution

Samples from a Gamma distribution.

The Gamma distribution is a continuous probability distribution that arises frequently in Bayesian statistics, particularly in prior distributions for variance parameters. It is defined by two positive shape parameters, concentration (k) and rate (theta).

```
\begin{align*}
f(x) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}, \quad x > 0
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

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

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

Example Usage:

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

## Methods

### GammaPoisson Distribution

A compound distribution comprising of a gamma-poisson pair, also referred to as a gamma-poisson mixture. The `rate` parameter for the :class:~`numpyro.distributions.Poisson` distribution is unknown and randomly drawn from a :class:~`numpyro.distributions.Gamma` distribution.

```
\begin{align*}
P(X = x) = \int_0^\infty \frac{1}{x} \exp(-x \lambda) \frac{1}{\Gamma(\alpha)} x^{\alpha - 1
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro 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 NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Gaussian Copula Distribution

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.

```
\begin{align*}
f(x_1, ..., x_d) = \prod_{i=1}^{d} f_i(x_i) \cdot \phi(F_1(x_1), ..., F_d(x_d); \mu, \Sigma)
\end{align*}
```

where: - f_i is the probability density function of the i-th marginal distribution. - F_i is the cumulative distribution function of the i-th marginal distribution. - ϕ is the standard normal PDF. - μ is the mean vector of the multivariate normal distribution. - Σ is the covariance matrix of the 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=0,
shape=(),
event=0,
create_obj=False,
)
```

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

NumPyro GaussianCopula distribution object: When `sample=False` (for model building). JAX array: When `sample=True` (for direct sampling). NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Gaussian Copula Beta distribution.

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.

```
\begin{align*}
f(x) = \int_{-\infty}^{\infty} g(x|u) h(u) du
\end{align*}
```

Where: - g(x|u) is the Gaussian copula density. - h(u) is the Beta density.

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

GaussianCopulaBeta: A NumPyro GaussianCopulaBeta distribution object (for model building). jnp.ndarray: A JAX array of samples drawn from the GaussianCopulaBeta distribution (for direct sampling). Distribution: The raw NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Gaussian Random Walk Distribution.

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.

```
\begin{align*}
X_t = X_{t-1} + \epsilon_t, \quad \text{where} \quad \epsilon_t \sim \mathcal{N}(0, \sigma^2
\end{align*}
```

with the initial state :math:`X_0 = 0`. The resulting sample is a vector of length `num_steps`, representing the path :math:`(X_1, X_2, \dots, X_{\text{num\_steps}})`.

Args:

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

- *sample* (float, jnp.ndarray, optional): The standard deviation (:math:`\sigma`) 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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro distribution object instead of creating a sample site. This is essential for building complex distributions like `MixtureSameFamily`. Defaults to `False`.

Returns:

numpyro.primitives.Messenger: A NumPyro 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 NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Gaussian State Space Distribution

Samples from a Gaussian state space model.

```
\begin{align*}
\mathbf{z}_{t} &= \mathbf{A} \mathbf{z}_{t - 1} + \boldsymbol{\epsilon}_t\\
&=\sum_{k=1} \mathbf{A}^{t-k} \boldsymbol{\epsilon}_t,
\end{align*}
```

where :math:`\mathbf{z}_t` is the state vector at step :math:`t`, :math:`\mathbf{A}` is the transition matrix, and :math:`\boldsymbol\epsilon` is the innovation noise.

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

num_steps (int): Number of steps.

transition_matrix (jnp.ndarray): State space transition matrix :math:`\mathbf{A}`.

covariance_matrix (jnp.ndarray, optional): Covariance of the innovation noise :math:`\boldsymbol{\epsilon}`. Defaults to None.

precision_matrix (jnp.ndarray, optional): Precision matrix of the innovation noise :math:`\boldsymbol{\epsilon}`. Defaults to None.

scale_tril (jnp.ndarray, optional): Scale matrix of the innovation noise :math:`\boldsymbol{\epsilon}`. 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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro 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 NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Geometric distribution.

The Geometric distribution models the number of failures before the first success in a sequence of Bernoulli trials. It is characterized by a single parameter, the probability of success on each trial.

```
\begin{align*}
P(X = k) = (1 - p)^k p
\end{align*}
```

Args:

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

- *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 `.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.
- *create_obj* (bool, optional): If True, returns the raw NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

```python
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

Sampled from distributions

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

## Methods

### GeometricLogits Distribution

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.

```
\begin{align*}
P(X = k) = (1 - p)^k p
\end{align*}
```

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

## Methods

### GeometricProbs

Samples from a Geometric distribution.

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.

```
\begin{align*}
P(X = k) = (1 - p)^k p
\end{align*}
```

Args:

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

- *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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

NumPyro GeometricProbs distribution object (for model building). JAX array of samples drawn from the GeometricProbs distribution (for direct sampling). The raw NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Gompertz Distribution.

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:

```
\begin{align*}
f(x) = \frac{con}{rate} \exp \left\{ - \frac{con}{rate} \left [ \exp\{x * rate \} - 1 \right
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

NumPyro Gompertz distribution object: When `sample=False` (for model building). JAX array: When `sample=True` (for direct sampling). NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### 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.

```
\begin{align*}
f(x) = \frac{1}{s} e^{-(x - \mu) / s} e^{-e^{- (x - \mu) / s}}
\end{align*}
```

Args:

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

- *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 `.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). Defaults to 1.
- *mask* (jnp.ndarray, bool, optional): Optional boolean array to mask observations. Defaults to None.
- *create_obj* (bool, optional): If True, returns the raw NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro 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 NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### HalfCauchy Distribution

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.

```
\begin{align*}
f(x) = \frac{1}{2} \cdot \frac{1}{\pi \cdot \frac{1}{scale} \cdot (x^2 + \frac{1}{scale^2})}
\end{align*}
```

Args:

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

- *sample* (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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro 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 NumPyro 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

Sampled from distributions

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

## Methods

### 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.

```
\begin{align*}
f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \text{ for } x > 0
\end{align*}
```

Args:

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

- *sample* (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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro 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 NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Improper Uniform Distribution

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

```
\begin{align*}
p(x) = 0
\end{align*}
```

Args:

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

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): If True, returns the raw NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

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

## Methods

### Inverse Gamma Distribution

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

```
\begin{align*}
p(x) = \frac{1}{Gamma(\alpha)} \left( \frac{\beta}{\Gamma(\alpha)} \right)^{\alpha} x^{\alpha
\text{ for } x > 0
\end{align*}
```

Args:

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

concentration (jnp.ndarray): The shape parameter (\alpha) of the InverseGamma distribution. Must be positive. rate (jnp.ndarray): The rate parameter (\beta) of the InverseGamma distribution. Must be positive.

- *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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

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

## Methods

### Kumaraswamy Distribution.

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.

```
\begin{align*}
f(x; a, b) = a b x^{a b - 1} (1 - x)^{b - 1}
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

```python
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

Sampled from distributions

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

## Methods

### Laplace Distribution

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).

```
\begin{align*}
f(x) = \frac{1}{2s} \exp\left(-\frac{|x - \mu|}{s}\right)
\end{align*}
```

Args:

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

- *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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

- When `sample=False`: A NumPyro 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 NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Left Truncated Distribution

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.

```
\begin{align*}
f(x) = \begin{cases}
\frac{f(x)}{P(X > \text{low})} & \text{if } x > \text{low} \\
0 & \text{otherwise}
\end{cases}
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

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

Sampled from distributions

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

## Methods

### Levy distribution.

Samples from a Levy distribution.

The probability density function is given by,

```
\begin{align*}
f(x\mid \mu, c) = \sqrt{\frac{c}{2\pi(x-\mu)^{3}}} \exp\left(-\frac{c}{2(x-\mu)}\right), \qqu
\end{align*}
```

where :math:`\mu` is the location parameter and :math:`c` is the scale parameter.

Args:

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

- *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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

NumPyro Levy distribution object: When `sample=False` (for model building). JAX array: When `sample=True` (for direct sampling). NumPyro 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 # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### Lewandowski Kurowicka Joe distribution

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

```
\begin{align*}
P(M) \propto |\det(M)|^{\eta - 1}
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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 NumPyro model. This is used to uniquely identify the random variable. Defaults to ‘x’.

Returns:

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

Example Usage:

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

## Methods

### LKJ Cholesky Distribution

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. \end{align*}

~~~
#### Args:

```python
bi.dist.lkj_cholesky(
dimension,
concentration=1.0,
sample_method='onion',
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=0,
~~~

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Attributes: concentration (float): The concentration parameter. # Sampled from distributions `Utils.np_dists.UnifiedDist` is a class to unify various distribution methods and provide a consistent interface for sampling and inference.

## Methods

### LogNormal distribution.

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

```
\begin{align*}
f(x) = \frac{1}{x \sigma \sqrt{2\pi}} e^{-\frac{(log(x) - \mu)^2}{2\sigma^2}}
\end{align*}
```

Args:

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

- *loc* (float): Location parameter.
- *sample* (float): Scale parameter.
- *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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Sampled from distributions

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

## Methods

### LogUniform

Samples from a LogUniform distribution.

The LogUniform 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.

```
\begin{align*}
f(x) = \frac{1}{(high - low) \log(high / low)}
\text{ for } low \le x \le high
\end{align*}
```

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

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

## Methods

### Logistic Distribution

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.

```
\begin{align*}
f(x) = \frac{1}{s} \exp\left(-\frac{(x - \mu)}{s}\right)
\end{align*}
```

Args:

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

- *loc* (jnp.ndarray or float): The location parameter, specifying the median of the distribution. Defaults to 0.0.
- *sample* (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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro model. This is used to uniquely identify the random variable. Defaults to ‘x’.

Returns:

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

## Methods

### Low Rank Multivariate Normal Distribution

Represents a multivariate normal distribution with a low-rank covariance structure.

```
\begin{align*}
p(x) = \frac{1}{\sqrt{(2\pi)^K |\Sigma|}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x
\end{align*}
```

where:

- :math:`x` is a vector of observations.
- :math:`\mu` is the mean vector.
- :math:`\Sigma` is the covariance matrix, represented in a low-rank form.

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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro 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

```
bi.dist.low_rank_multivariate_normal(
loc,
cov_factor,
cov_diag,
validate_args=None,
name='x',
obs=None,
mask=None,
sample=False,
seed=0,
shape=(),
event=0,
create_obj=False,
)
```

Sampled from distributions

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

## Methods

### LowerTruncatedPowerLaw

Lower truncated power law distribution with :math:`\alpha` index.

The probability density function (PDF) is given by:

```
\begin{align*}
f(x; \alpha, a) = (-\alpha-1)a^{-\alpha - 1}x^{-\alpha},
\qquad x \geq a, \qquad \alpha < -1,
\end{align*}
```

where :math:`a` is the lower bound.

Args:

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

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 `.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 NumPyro 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 NumPyro model context. If `False`, it will create a `numpyro.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`. [7] This argument has no effect when `sample=False`, as randomness is handled by NumPyro'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 NumPyro model. This is used to uniquely identify the random variable. Defaults to 'x'.

Returns:

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

Example Usage:

```python
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