Package ‘ROOPSD’

January 20, 2025

Title R Object Oriented Programming for Statistical Distribution

Date 2023-09-11

Version 0.3.9

Description Statistical distribution in OOP (Object Oriented Programming) way.
This package proposes a R6 class interface to classic statistical distribution, and new distributions can be easily added with the class AbstractDist. A useful point is the generic fit() method for each class, which uses a maximum likelihood estimation to find the parameters of a dataset, see, e.g. Hastie, T. and al (2009) <isbn:978-0-387-84857-0>. Furthermore, the rv_histogram class gives a non-parametric fit, with the same accessors that for the classic distribution. Finally, three random generators useful to build synthetic data are given: a multivariate normal generator, an orthogonal matrix generator, and a symmetric positive definite matrix generator, see Mezzadri, F. (2007) <arXiv:math-ph/0609050>.

URL https://github.com/yrobink/ROOPSD

Depends R (>= 3.3)

License CeCILL-2

Encoding UTF-8

Imports methods, R6, Lmoments, numDeriv

RoxygenNote 7.2.3

NeedsCompilation no

Author Yoann Robin [aut, cre]

Maintainer Yoann Robin <yoann.robin.k@gmail.com>

Repository CRAN

Date/Publication 2023-09-11 07:20:15 UTC

Contents

`AbstractDist` *AbstractDist*

Description

Base class for OOP statistical distribution

Details

This class is only used to be herited

Public fields

`ddist` [function] density function

`pdist` [function] distribution function

`qdist` [function] quantile function

`rdist` [function] random generator function

`ks.test` [ks.test] Goodness of fit with ks.test

`fit_success` [bool] TRUE only if the fit is a success and is occurred

Active bindings

`name` [string] name of the distribution

`opt` [stats::optim result] Result of the MLE to find parameters

`cov` [matrix] Covariance matrix of parameters, inverse of hessian

`coef` [vector] Vector of coefficients

Methods

Public methods:

- `AbstractDist$new()`
- `AbstractDist$rvs()`
- `AbstractDist$density()`
- `AbstractDist$logdensity()`
- `AbstractDist$cdf()`
- `AbstractDist$sf()`
- `AbstractDist$icdf()`
- `AbstractDist$isf()`
- `AbstractDist$fit()`
- `AbstractDist$qgradient()`
- `AbstractDist$qdeltaCI()`
- `AbstractDist$pdeltaCI()`
- `AbstractDist$diagnostic()`
- `AbstractDist$clone()`

Method `new()`**:** Create a new AbstractDist object.

Usage:

```
AbstractDist$new(ddist, pdist, qdist, rdist, name, has_gr_nlll)
```

Arguments:

`ddist` [function] Density function, e.g. dnorm

`pdist` [function] Distribution function, e.g. pnorm

`qdist` [function] Quantile function, e.g. qnorm

`rdist` [function] Random generator function, e.g. rnorm

`name` [str] name of the distribution

`has_gr_nlll` [bool] If the derived class has defined the gradient of the negative log-likelihood

Returns: A new ‘AbstractDist‘ object.

Method `rvs()`**:** Generation sample from the histogram

Usage:

```
AbstractDist$rvs(n)
```

Arguments:

`n` [integer] Number of samples drawn

Returns: [vector] A vector of samples

Method `density()`**:** Density function

Usage:

`AbstractDist$density(x)`

Arguments:

`x` [vector] Values to compute the density

Returns: [vector] density

Method `logdensity()`**:** Log density function

Usage:

`AbstractDist$logdensity(x)`

Arguments:

`x` [vector] Values to compute the log-density

Returns: [vector] log of density

Method `cdf()`**:** Cumulative Distribution Function

Usage:

`AbstractDist$cdf(q)`

Arguments:

`q` [vector] Quantiles to compute the CDF

Returns: [vector] cdf values

Method `sf()`**:** Survival Function

Usage:

`AbstractDist$sf(q)`

Arguments:

`q` [vector] Quantiles to compute the SF

Returns: [vector] sf values

Method `icdf()`**:** Inverse of Cumulative Distribution Function

Usage:

`AbstractDist$icdf(p)`

Arguments:

`p` [vector] Probabilities to compute the CDF

Returns: [vector] icdf values

Method `isf()`**:** Inverse of Survival Function

Usage:

`AbstractDist$isf(p)`

Arguments:

`p` [vector] Probabilities to compute the SF

Returns: [vector] isf values

Method `fit()`**:** Fit method

Usage:

`AbstractDist$fit(Y, n_max_try = 100)`

Arguments:

`Y` [vector] Dataset to infer the histogram

`n_max_try` [integer] Because the optim function can fails, the fit is retry n_try times.

Returns: ‘self‘

Method `qgradient()`**:** Gradient of the quantile function

Usage:

`AbstractDist$qgradient(p, lower.tail = TRUE)`

Arguments:

`p` [vector] Probabilities

`lower.tail` [bool] If CDF or SF.

Returns: [vector] gradient

Method `qdeltaCI()`**:** Confidence interval of the quantile function

Usage:

`AbstractDist$qdeltaCI(p, Rt = FALSE, alpha = 0.05)`

Arguments:

`p` [vector] Probabilities

`Rt` [bool] if Probabilities or return times

`alpha` [double] level of confidence interval

Returns: [list] Quantiles, and confidence interval

Method `pdeltaCI()`**:** Confidence interval of the CDF function

Usage:

`AbstractDist$pdeltaCI(x, Rt = FALSE, alpha = 0.05)`

Arguments:

`x` [vector] Quantiles

`Rt` [bool] if Probabilities or return times

`alpha` [double] level of confidence interval

Returns: [list] CDF, and confidence interval

Method `diagnostic()`**:** Diagnostic of the fitted law

Usage:

`AbstractDist$diagnostic(Y, alpha = 0.05)`

Arguments:

`Y` [vector] data to check

`alpha` [double] level of confidence interval

Returns: [NULL]

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
AbstractDist$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`dgev` *dgev*

Description

Density function of Generalized Extreme Value distribution

Usage

```
dgev(x, loc = 0, scale = 1, shape = 0, log = FALSE)
```

Arguments

`x`	[vector] Vector of values
`loc`	[vector] Location parameter
`scale`	[vector] Scale parameter
`shape`	[vector] Shape parameter
`log`	[bool] Return log of density if TRUE, default is FALSE

Value

[vector] Density of GEV at x

Examples

```
## Data
loc = 1
scale = 0.5
shape = -0.2
x = base::seq( -5 , 5 , length = 1000 )
y = dgev( x , loc = loc , scale = scale , shape = shape )
```

---

dgpd *dgpd*

Description

Density function of Generalized Pareto Distribution

Usage

```
dgpd(x, loc = 0, scale = 1, shape = 0, log = FALSE)
```

Arguments

`x`	[vector] Vector of values
`loc`	[vector] Location parameter
`scale`	[vector] Scale parameter
`shape`	[vector] Shape parameter
`log`	[bool] Return log of density if TRUE, default is FALSE

Value

[vector] Density of GPD at x

Examples

```
## Data
loc = 1
scale = 0.5
shape = -0.2
x = base::seq( -5 , 5 , length = 1000 )
y = dgpd( x , loc = loc , scale = scale , shape = shape )
```

Exponential *Exponential*

Description

Exponential distribution in OOP way. Based on AbstractDist

Details

See AbstractDist for generic methods

Super class

`ROOPSD::AbstractDist` -> `Exponential`

Active bindings

`rate` [double] rate of the exponential law

`params` [vector] params of the exponential law

Methods

Public methods:

- `Exponential$new()`
- `Exponential$clone()`

Method `new()`**:** Create a new Exponential object.

Usage:

```
Exponential$new(rate = 1)
```

Arguments:

`rate` [double] Rate of the exponential law

Returns: A new ‘Exponential‘ object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Exponential$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Generate sample
rate = 0.5
expl = ROOPSD::Exponential$new( rate = rate )
X    = expl$rvs( n = 1000 )

## And fit parameters
expl$fit(X)
```

Gamma *Gamma*

Description

Gamma distribution in OOP way. Based on AbstractDist

Details

See AbstractDist for generic methods

Super class

`ROOPSD::AbstractDist` -> `Gamma`

Active bindings

`shape` [double] shape of the gamma law

`scale` [double] scale of the gamma law

`params` [vector] params of the gamma law

Methods

Public methods:

- `Gamma$new()`
- `Gamma$clone()`

Method `new()`**:** Create a new Gamma object.

Usage:

```
Gamma$new(shape = 0.5, scale = 1)
```

Arguments:

`shape` [double] shape parameter

`scale` [double] scale parameter

Returns: A new ‘Gamma‘ object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Gamma$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Generate sample
scale = 1.5
shape = 0.5
gaml = ROOPSD::Gamma$new( scale = scale , shape = shape )
X    = gaml$rvs( n = 1000 )

## And fit parameters
gaml$fit(X)
```

GEV *GEV*

Description

GEV distribution in OOP way. Based on AbstractDist

Details

See AbstractDist for generic methods

Super class

`ROOPSD::AbstractDist` -> `GEV`

Active bindings

`loc` [double] location of the GEV law

`scale` [double] scale of the GEV law

`shape` [double] shape of the GEV law

`params` [vector] params of the GEV law

Methods

Public methods:

- `GEV$new()`
- `GEV$qgradient()`
- `GEV$pgradient()`
- `GEV$clone()`

Method `new()`**:** Create a new GEV object.

Usage:

```
GEV$new(loc = 0, scale = 1, shape = -0.1)
```

Arguments:

`loc` [double] location parameter

`scale` [double] scale parameter

`shape` [double] shape parameter

Returns: A new ‘GEV‘ object.

Method `qgradient()`**:** Gradient of the quantile function

Usage:

```
GEV$qgradient(p, lower.tail = TRUE)
```

Arguments:

p [vector] Probabilities
lower.tail [bool] If CDF or SF.

Returns: [vector] gradient

Method `pgradient()`**:** Gradient of the CDF function

Usage:

```
GEV$pgradient(x, lower.tail = TRUE)
```

Arguments:

x [vector] Quantiles
lower.tail [bool] If CDF or SF.

Returns: [vector] gradient

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GEV$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## Generate sample
loc   = 0
scale = 0.5
shape = -0.3
gev = ROOPSD::GEV$new( loc = loc , scale = scale , shape = shape )
X   = gev$rvs( n = 1000 )

## And fit parameters
gev$fit(X)
```

GPD *GPD*

Description

GPD distribution in OOP way. Based on AbstractDist

Details

See AbstractDist for generic methods

Super class

ROOPSD::AbstractDist -> GPD

Active bindings

`loc` [double] location of the GPD law, fixed

`scale` [double] scale of the GPD law

`shape` [double] shape of the GPD law

`params` [vector] params of the GPD law

Methods

Public methods:

- `GPD$new()`
- `GPD$fit()`
- `GPD$clone()`

Method `new()`**:** Create a new GPD object.

Usage:

```
GPD$new(loc = 0, scale = 1, shape = -0.1)
```

Arguments:

`loc` [double] location parameter

`scale` [double] scale parameter

`shape` [double] shape parameter

Returns: A new 'GPD' object.

Method `fit()`**:** Fit method

Usage:

```
GPD$fit(Y, loc = NULL)
```

Arguments:

`Y` [vector] Dataset to infer the histogram

`loc` [double] location parameter, if NULL used min(Y)

Returns: 'self'

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GPD$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## Generate sample
loc   = 0
scale = 0.5
shape = -0.3
gpd = ROOPSD::GPD$new( loc = loc , scale = scale , shape = shape )
```

```
X   = gpd$rvs( n = 1000 )

## And fit parameters
gpd$fit( X , loc = 0 )
```

`mrv_histogram` *mrv_histogram*

Description

Multivariate rv_histogram distribution in OOP way.

Details

Used for a multivariate dataset, fit each marge

Public fields

`n_features` [integer] Number of features (dimensions)

`law_` [list] List of marginal distributions

Methods

Public methods:

- `mrv_histogram$new()`
- `mrv_histogram$fit()`
- `mrv_histogram$rvs()`
- `mrv_histogram$cdf()`
- `mrv_histogram$sf()`
- `mrv_histogram$icdf()`
- `mrv_histogram$isf()`
- `mrv_histogram$clone()`

Method `new()`**:** Create a new mrv_histogram object.

Usage:

```
mrv_histogram$new(...)
```

Arguments:

`...` If a param 'Y' is given, the fit method is called with '...'.

Returns: A new 'mrv_histogram' object.

Method `fit()`**:** Fit method for the histograms

Usage:

```
mrv_histogram$fit(Y, bins = as.integer(100))
```

Arguments:

`Y` [vector] Dataset to infer the histogram

`bins` [list or vector or integer] bins values

Returns: 'self'

Method `rvs()`**:** Generation sample from the histogram

Usage:

`mrv_histogram$rvs(n = 1)`

Arguments:

`n` [integer] Number of samples drawn

Returns: A matrix of samples

Method `cdf()`**:** Cumulative Distribution Function

Usage:

`mrv_histogram$cdf(q)`

Arguments:

`q` [vector] Quantiles to compute the CDF

Returns: cdf values

Method `sf()`**:** Survival Function

Usage:

`mrv_histogram$sf(q)`

Arguments:

`q` [vector] Quantiles to compute the SF

Returns: sf values

Method `icdf()`**:** Inverse of Cumulative Distribution Function

Usage:

`mrv_histogram$icdf(p)`

Arguments:

`p` [vector] Probabilities to compute the CDF

Returns: icdf values

Method `isf()`**:** Inverse of Survival Function

Usage:

`mrv_histogram$isf(p)`

Arguments:

`p` [vector] Probabilities to compute the SF

Returns: isf values

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`mrv_histogram$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Generate sample
X = matrix( stats::rnorm( n = 10000 ) , ncol = 4 )

## And fit it
rvX = mrv_histogram$new()
rvX$fit(X)
```

`Normal` *Normal*

Description

Normal distribution in OOP way. Based on AbstractDist

Details

See AbstractDist for generic methods

Super class

`ROOPSD::AbstractDist` -> `Normal`

Active bindings

`mean` [double] mean of the normal law

`sd` [double] standard deviation of the normal law

`params` [vector] params of the normal law

Methods

Public methods:

- `Normal$new()`
- `Normal$clone()`

Method `new()`: Create a new Normal object.

Usage:

```
Normal$new(mean = 0, sd = 1)
```

Arguments:

`mean` [double] Mean of the normal law

`sd` [double] Standard deviation of the normal law

Returns: A new ‘Normal‘ object.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`Normal$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Generate sample
mean  = 1
sd    = 0.5
norml = ROOPSD::Normal$new( mean = mean , sd = sd )
X     = norml$rvs( n = 1000 )

## And fit parameters
norml$fit(X)
```

---

pgev *pgev*

Description

Cumulative distribution function (or survival function) of Generalized Extreme Value distribution

Usage

```
pgev(q, loc = 0, scale = 1, shape = 0, lower.tail = TRUE)
```

Arguments

`q`	[vector] Vector of quantiles
`loc`	[vector] Location parameter
`scale`	[vector] Scale parameter
`shape`	[vector] Shape parameter
`lower.tail`	[bool] Return CDF if TRUE, else return survival function

Value

[vector] CDF (or SF) of GEV at x

Examples

```
## Data
loc = 1
scale = 0.5
shape = -0.2
x = base::seq( -5 , 5 , length = 1000 )
cdfx = pgev( x , loc = loc , scale = scale , shape = shape )
```

pgpd — *pgpd*

Description

Cumulative distribution function (or survival function) of Generalized Pareto distribution

Usage

```
pgpd(q, loc = 0, scale = 1, shape = 0, lower.tail = TRUE)
```

Arguments

`q`	[vector] Vector of quantiles
`loc`	[vector] Location parameter
`scale`	[vector] Scale parameter
`shape`	[vector] Shape parameter
`lower.tail`	[bool] Return CDF if TRUE, else return survival function

Value

[vector] CDF (or SF) of GPD at x

Examples

```
## Data
loc = 1
scale = 0.5
shape = -0.2
x = base::seq( -5 , 5 , length = 1000 )
cdfx = pgpd( x , loc = loc , scale = scale , shape = shape )
```

qgev — *qgev*

Description

Inverse of CDF (or SF) function of Generalized Extreme Value distribution

Usage

```
qgev(p, loc = 0, scale = 1, shape = 0, lower.tail = TRUE)
```

Arguments

`p`	[vector] Vector of probabilities
`loc`	[vector] Location parameter
`scale`	[vector] Scale parameter
`shape`	[vector] Shape parameter
`lower.tail`	[bool] Return inverse of CDF if TRUE, else return inverse of survival function

Value

[vector] Inverse of CDF or SF of GEV for probabilities p

Examples

```
## Data
loc = 1
scale = 0.5
shape = -0.2
p = base::seq( 0.01 , 0.99 , length = 100 )
q = qgev( p , loc = loc , scale = scale , shape = shape )
```

---

qgpd *qgpd*

Description

Inverse of CDF (or SF) function of Generalized Pareto distribution

Usage

```
qgpd(p, loc = 0, scale = 1, shape = 0, lower.tail = TRUE)
```

Arguments

`p`	[vector] Vector of probabilities
`loc`	[vector] Location parameter
`scale`	[vector] Scale parameter
`shape`	[vector] Shape parameter
`lower.tail`	[bool] Return inverse of CDF if TRUE, else return inverse of survival function

Value

[vector] Inverse of CDF or SF of GPD for probabilities p

Examples

```
## Data
loc = 1
scale = 0.5
shape = -0.2
p = base::seq( 0.01 , 0.99 , length = 100 )
q = qgpd( p , loc = loc , scale = scale , shape = shape )
```

---

rgev *rgev*

Description

Random value generator of Generalized Extreme Value distribution

Usage

```
rgev(n = 1, loc = 0, scale = 1, shape = 0)
```

Arguments

`n`	[int] Numbers of values generated
`loc`	[vector] Location parameter
`scale`	[vector] Scale parameter
`shape`	[vector] Shape parameter

Value

[vector] Random value following a GEV(loc,scale,shape)

Examples

```
## Data
loc = 1
scale = 0.5
shape = -0.2
gev = rgev( 100 , loc = loc , scale = scale , shape = shape )
```

rgpd *rgpd*

Description

Random value generator of Generalized Pareto distribution

Usage

```
rgpd(n = 1, loc = 0, scale = 1, shape = 0)
```

Arguments

n	[int] Numbers of values generated
loc	[vector] Location parameter
scale	[vector] Scale parameter
shape	[vector] Shape parameter

Value

[vector] Random value following a loc + GPD(scale,shape)

Examples

```
## Data
loc = 1
scale = 0.5
shape = -0.2
gev = rgpd( 100 , loc = loc , scale = scale , shape = shape )
```

rmultivariate_normal *rmultivariate_normal*

Description

Generate sample from a multivariate normal distribution. The generator uses a singular values decomposition to draw samples from a normal distribution in the basis of the singular vector. Consequently, the covariance matrix can be singular.

Usage

```
rmultivariate_normal(n, mean, cov)
```

Arguments

`n`	[integer] numbers of samples drawn
`mean`	[vector] mean of Normal law
`cov`	[matrix] covariance matrix

Value

[matrix]

Examples

```
mean = stats::runif( n = 2 , min = -5 , max = 5 )
cov  = ROOPSD::rspd_matrix(2)
X    = ROOPSD::rmultivariate_normal( 10000 , mean , cov )
```

`rorthogonal_group` *rorthogonal_group*

Description

Generate sample from the orthogonal group O(d)

Usage

```
rorthogonal_group(d, n = 1)
```

Arguments

`d`	[integer] Dimension of the matrix
`n`	[integer] numbers of samples drawn

Value

[array or matrix], dim = d * d * n or d * d if n == 1

Examples

```
M = ROOPSD::rorthogonal_group( 2 , 10 )
```

rspd_matrix *rspd_matrix*

Description

Generate a random symetric positive definite matrix. The generator just draw matrix of the form O * diag(positive values) * t(O), where O is an orthogonal matrix from ROOPSD::rorthogonal_group. Note that the parameter gen = stats::rexp draw positive eigen values, but the code do not control if eigen values are positive. So you can accept negative eigen values using another generators.

Usage

```
rspd_matrix(d, n = 1, sort_eigenvalues = TRUE, gen = stats::rexp)
```

Arguments

d	[integer] Dimension of the matrix
n	[integer] numbers of samples drawn
sort_eigenvalues	[bool] If eigen values (i.e. variance) are sorted
gen	[function] Eigenvalues generator

Value

[array or matrix], dim = d * d * n or d * d if n == 1

Examples

```
mean = stats::runif( n = 2 , min = -5 , max = 5 )
cov  = ROOPSD::rspd_matrix(2)
X    = ROOPSD::rmultivariate_normal( 10000 , mean , cov )
```

rv_histogram *rv_histogram*

Description

rv_histogram distribution in OOP way.

Details

Use quantile to fit the histogram

Public fields

`min` [double] min value for the estimation

`max` [double] max value for the estimation

`tol` [double] numerical tolerance

Methods

Public methods:

- `rv_histogram$new()`
- `rv_histogram$rvs()`
- `rv_histogram$density()`
- `rv_histogram$logdensity()`
- `rv_histogram$cdf()`
- `rv_histogram$icdf()`
- `rv_histogram$sf()`
- `rv_histogram$isf()`
- `rv_histogram$fit()`
- `rv_histogram$clone()`

Method `new()`**:** Create a new rv_histogram object.

Usage:

```
rv_histogram$new(...)
```

Arguments:

`...` If a param ‘Y‘ is given, the fit method is called with ‘...‘.

Returns: A new ‘rv_histogram‘ object.

Method `rvs()`**:** Generation sample from the histogram

Usage:

```
rv_histogram$rvs(n)
```

Arguments:

`n` [integer] Number of samples drawn

Returns: A vector of samples

Method `density()`**:** Density function

Usage:

```
rv_histogram$density(x)
```

Arguments:

`x` [vector] Values to compute the density

Returns: density

Method `logdensity()`**:** Log density function

Usage:

`rv_histogram$logdensity(x)`

Arguments:

`x` [vector] Values to compute the log-density

Returns: the log density

Method `cdf()`**:** Cumulative Distribution Function

Usage:

`rv_histogram$cdf(q)`

Arguments:

`q` [vector] Quantiles to compute the CDF

Returns: cdf values

Method `icdf()`**:** Inverse of Cumulative Distribution Function

Usage:

`rv_histogram$icdf(p)`

Arguments:

`p` [vector] Probabilities to compute the CDF

Returns: icdf values

Method `sf()`**:** Survival Function

Usage:

`rv_histogram$sf(q)`

Arguments:

`q` [vector] Quantiles to compute the SF

Returns: sf values

Method `isf()`**:** Inverse of Survival Function

Usage:

`rv_histogram$isf(p)`

Arguments:

`p` [vector] Probabilities to compute the SF

Returns: isf values

Method `fit()`**:** Fit method for the histograms

Usage:

`rv_histogram$fit(Y, bins = as.integer(1000))`

Arguments:

`Y` [vector] Dataset to infer the histogram

`bins` [vector or integer] bins values

Returns: ‘self‘

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`rv_histogram$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Generate sample
X = numeric(10000)
X[1:5000] = stats::rnorm( n = 5000 , mean = 2 , sd = 1 )
X[5000:10000] = stats::rexp( n = 5000 , rate = 1 )

## And fit it
rvX = rv_histogram$new()
rvX$fit(X)
```

`rv_mixture` *rv_mixture*

Description

rv_mixture distribution in OOP way.

Details

No fit allowed.

Active bindings

`l_dist` [list] List of distributions.

`n_dist` [integer] Numbers of distribution.

`weights` [vector] Weights of the distributions.

Methods

Public methods:

- `rv_mixture$new()`
- `rv_mixture$rvs()`
- `rv_mixture$density()`
- `rv_mixture$logdensity()`
- `rv_mixture$cdf()`
- `rv_mixture$icdf()`
- `rv_mixture$sf()`
- `rv_mixture$isf()`
- `rv_mixture$clone()`

Method `new()`**:** Create a new rv_mixture object.

Usage:

```
rv_mixture$new(l_dist, weights = NULL)
```

Arguments:

`l_dist` [list] List of ROOPSD distributions.

`weights` [vector] Weights of the distributions. If NULL, 1 / length(l_dist) is used.

Returns: A new ‘rv_mixture‘ object.

Method `rvs()`**:** Generation sample from the histogram

Usage:

```
rv_mixture$rvs(n)
```

Arguments:

`n` [integer] Number of samples drawn

Returns: A vector of samples

Method `density()`**:** Density function

Usage:

```
rv_mixture$density(x)
```

Arguments:

`x` [vector] Values to compute the density

Returns: density

Method `logdensity()`**:** Log density function

Usage:

```
rv_mixture$logdensity(x)
```

Arguments:

`x` [vector] Values to compute the log-density

Returns: the log density

Method `cdf()`**:** Cumulative Distribution Function

Usage:

```
rv_mixture$cdf(q)
```

Arguments:

`q` [vector] Quantiles to compute the CDF

Returns: cdf values

Method `icdf()`**:** Inverse of Cumulative Distribution Function

Usage:

```
rv_mixture$icdf(p)
```

Arguments:

`p` [vector] Probabilities to compute the CDF

Returns: icdf values

Method `sf()`**:** Survival Function

Usage:

```
rv_mixture$sf(q)
```

Arguments:

`q` [vector] Quantiles to compute the SF

Returns: sf values

Method `isf()`**:** Inverse of Survival Function

Usage:

```
rv_mixture$isf(p)
```

Arguments:

`p` [vector] Probabilities to compute the SF

Returns: isf values

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
rv_mixture$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Define the mixture
l_dist  = list( Exponential$new() , Normal$new( mean = 5 , sd = 1 ) )
weights = base::c( 0.2 , 0.8 )
rvX = rv_mixture$new( l_dist , weights )

## Draw samples
X   = rvX$rvs( 1000 )
```

`rv_ratio_histogram` *rv_ratio_histogram*

Description

rv_ratio_histogram distribution in OOP way.

Details

Fit separatly P(X < x | X > 0) and P(X=0)

Public fields

`rvXp` [ROOPSD::rv_histogram] Describes P(X < x | X > x0)

`x0` [double] location of mass: P(X = x0)

`p0` [double] p0 = P(X = x0)

Methods

Public methods:

- rv_ratio_histogram$new()
- rv_ratio_histogram$rvs()
- rv_ratio_histogram$cdf()
- rv_ratio_histogram$icdf()
- rv_ratio_histogram$sf()
- rv_ratio_histogram$isf()
- rv_ratio_histogram$fit()
- rv_ratio_histogram$clone()

Method `new()`**:** Create a new rv_ratio_histogram object.

Usage:

```
rv_ratio_histogram$new(...)
```

Arguments:

`...` If a param ‘Y‘ and ‘x0‘ is given, the fit method is called with ‘...‘.

Returns: A new ‘rv_ratio_histogram‘ object.

Method `rvs()`**:** Generation sample from the histogram

Usage:

```
rv_ratio_histogram$rvs(n)
```

Arguments:

`n` [integer] Number of samples drawn

Returns: A vector of samples

Method `cdf()`**:** Cumulative Distribution Function

Usage:

```
rv_ratio_histogram$cdf(q)
```

Arguments:

`q` [vector] Quantiles to compute the CDF

Returns: cdf values

Method `icdf()`**:** Inverse of Cumulative Distribution Function

Usage:

```
rv_ratio_histogram$icdf(p)
```

Arguments:

`p` [vector] Probabilities to compute the CDF

Returns: icdf values

Method `sf()`**:** Survival Function

Usage:

```
rv_ratio_histogram$sf(q)
```

Arguments:

`q` [vector] Quantiles to compute the SF

Returns: sf values

Method `isf()`**:** Inverse of Survival Function

Usage:

```
rv_ratio_histogram$isf(p)
```

Arguments:

`p` [vector] Probabilities to compute the SF

Returns: isf values

Method `fit()`**:** Fit method for the histograms

Usage:

```
rv_ratio_histogram$fit(Y, x0, bins = as.integer(100))
```

Arguments:

`Y` [vector] Dataset to infer the histogram

`x0` [double] Location of mass point

`bins` [vector or integer] bins values

Returns: ‘self‘

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
rv_ratio_histogram$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Generate sample
X = numeric(10000)
X[1:2000] = 0
X[2001:10000] = stats::rexp( n = 8000 , rate = 1 )

## And fit it
rvX = rv_ratio_histogram$new()
rvX$fit( X , x0 = 0 )
```

`Uniform` *Uniform*

Description

Uniform distribution in OOP way. Based on AbstractDist

Details

See AbstractDist for generic methods

Super class

`ROOPSD::AbstractDist` -> `Uniform`

Active bindings

`min` [double] min of the uniform law

`max` [double] max of the uniform law

`params` [vector] params of the uniform law

Methods

Public methods:

- `Uniform$new()`
- `Uniform$clone()`

Method `new()`**:** Create a new Uniform object.

Usage:

```
Uniform$new(min = 0, max = 1)
```

Arguments:

`min` [double] Min of the uniform law

`max` [double] Max of the uniform law

Returns: A new ‘Uniform‘ object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Uniform$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Generate sample
min = -1
max = 1
unifl = ROOPSD::Uniform$new( min = min , max = max )
X     = unifl$rvs( n = 1000 )

## And fit parameters
unifl$fit(X)
```

Index