Package ‘poweRlaw’

August 11, 2015

Type Package
Title Analysis of Heavy Tailed Distributions
Version 0.50.0
Date 2015-08-12
Maintainer Colin Gillespie <csgillespie@gmail.com>

Description An implementation of maximum likelihood estimators for a variety of heavy tailed distributions, including both the discrete and continuous power law distributions. Additionally, a goodness-of-fit based approach is used to estimate the lower cut-off for the scaling region.

URL https://github.com/csgillespie/poweRlaw

BugReports https://github.com/csgillespie/poweRlaw/issues

Depends R (>= 3.0.0)
Imports VGAM, parallel, methods, utils, stats
Suggests knitr, R.matlab

ByteCompile true
VignetteBuilder knitr
License GPL-2 | GPL-3

Collate 'aaa_all_classes.R' 'AllGenerics.R' 'bootstrap.R'
'bootstrap_p.R' 'checks.R' 'compare_distributions.R'
'ctn_helper_functions.R' 'data_help_files.R' 'def_conexp.R'
'def_conlorm.R' 'def_conpl.R' 'def_disexp.R' 'def_dislnorm.R'
'def_displ.R' 'def_dispois.R' 'def_template.R'
'dist_data_cdf_methods.R' 'estimate_pars.R' 'estimate_xmin.R'
'get_n.R' 'get_ntail.R' 'lines_methods.R' 'plcon.R' 'pldis.R'
'plot_methods.R' 'points_methods.R' 'poweRlaw-package.R'
'show_methods.R'

NeedsCompilation no

Author Colin Gillespie [aut, cre]

Repository CRAN

Date/Publication 2015-08-11 15:18:32
Description

The poweRlaw package aims to make fitting power laws and other heavy-tailed distributions straightforward. This package contains R functions for fitting, comparing and visualising heavy tailed distributions. Overall, it provides a principled approach to fitting power laws to data.

Details

The code developed in this package has been heavily influenced from the python and R code found at: http://tuvalu.santafe.edu/~aaronc/powerlaws/. In particular, the R code of Laurent Dubroca and Cosma Shalizi.

Author(s)

<colin.gillespie@newcastle.ac.uk>
References

See Also
https://github.com/csgillespie/powerlaw

**Description**
When fitting heavy tailed distributions, sometimes it is necessary to estimate the lower threshold, xmin. The lower bound is estimated by calculating the minimising the Kolmogorov-Smirnoff statistic (as described in Clauset, Shalizi, Newman (2009)).

get_ks_statistic Calculates the KS statistic for a particular value of xmin.
estimate_xmin Estimates the optimal lower cutoff using a goodness-of-fit based approach. This function may issue warnings when fitting lognormal, Poisson or Exponential distributions. The warnings occur for large values of xmin. Essentially, we are discarding the bulk of the distribution and cannot calculate the tails to enough accuracy.

**Usage**
bootstrap(m, xmins = NULL, pars = NULL, xmax = 1e+05, no_of_sims = 100, threads = 1, seed = NULL)
bootstrap_p(m, xmins = NULL, pars = NULL, xmax = 1e+05, no_of_sims = 100, threads = 1, seed = NULL)
get_ks_statistic(m, xmax = 1e+05)
estimate_xmin(m, xmins = NULL, pars = NULL, xmax = 1e+05)

**Arguments**
m A reference class object that contains the data.
xmins default 1e5. A vector of possible values of xmin to explore. When a single value is passed, this represents the maximum value to search, i.e. by default we search from (1, 1e5). See details for further information.
pars  default NULL. A vector of parameters used to optimise over. Otherwise, for each value of xmin, the mle will be used, i.e. estimate_pars(m). For small samples, the mle may be biased.

xmax  default 1e5. The maximum x value calculated when working out the CDF. See details for further information.

no_of_sims  number of bootstrap simulations. When no_of_sims is large, this can take a while to run.

threads  number of concurrent threads used during the bootstrap.

seed  default NULL. An integer to be supplied to set.seed, or NULL not to set reproducible seeds. This argument is passed to clusterSetRNGStream.

Details

When estimating xmin for discrete distributions, the search space when comparing the data-cdf (empirical cdf) and the distribution_cdf runs from xmin to max(x) where x is the data set. This can often be computationally brutal. In particular, when bootstrapping we generate random numbers from the power law distribution, which has a long tail.

To speed up computations for discrete distributions it is sensible to put an upper bound, i.e. xmax or explicitly give values of where to search, i.e. xmin. When calculating the kolmogorov-smirnov, the CDF is calculated up to the maximum value of the data. However, for some data sets and during bootstrapping, very large values can be generated. This may cause both speed and memory problems.

Occasionally bootstrapping can generate strange situations. For example, all values in the simulated data set are less than xmin. In this case, the estimated KS statistic will be Inf and the parameter values, NA.

Note

Adapted from Laurent Dubroca’s code found at http://tuvalu.santafe.edu/~aaronc/powerlaws/plfit.r

Examples

```r
# Load the data set and create distribution object
x = 1:10
m = displ$new(x)

# Estimate xmin and pars
est = estimate_xmin(m)
m$setXmin(est)

# Bootstrap examples
## Not run:
```
**bootstrap_moby**

```r
bootstrap(m, no_of_sims=1, threads=1)
bootstrap_p(m, no_of_sims=1, threads=1)
```

## End(Not run)

---

**bootstrap_moby**  
*Example bootstrap results for the full Moby Dick data set*

### Description

To explore the uncertainty in the model fit, this package provides a `bootstrap` function.

- **bootstrap_moby**  
The output from running 5000 bootstraps on the full Moby Dick data set (for a discrete power law) using the `bootstrap` function.

- **bootstrap_p_moby**  
The output from running 5000 bootstraps on the full Moby Dick data set (for a discrete power law) using the `bootstrap_p` function.

The `bootstrap_moby` values correspond to the first row of table 6.1 in the Clauset et al paper:

```r
bootstrap_moby$gof  
```

- **bootstrap_moby$gof**  
The K-S statistic

```r
bootstrap_moby$bootstraps  
a data frame for the optimal values from the bootstrapping procedure. Column 1: K-S, Column 2: $x_{min}$, Column 3: $\alpha$. So standard deviation of column 2 and 3 is 2.2 and 0.033 (the paper gives 2 and 0.02 respectively).
```

The `bootstrap_p_moby` gives the p-value for the hypothesis test of whether the data follows a power-law. For this simulation study, we get a value of 0.43 (the paper gives 0.49).

### Format

A list

### Source


### See Also

- `moby`
- `bootstrap`
- `bootstrap_p`

### Examples

```r
## Generate the bootstrap_moby data set
## Not run:
data(moby)
m = displ$new(moby)
bs = bootstrap(m, no_of_sims=5000, threads=4, seed=1)

## End(Not run)
```
compare_distributions

Vuong’s test for non-nested models

Description

Since it is possible to fit power law models to any data set, it is recommended that alternative
distributions are considered. A standard technique is to use Vuong’s test. This is a likelihood ratio
test for model selection using the Kullback-Leibler criteria. The test statistic, R, is the ratio of
the log-likelihoods of the data between the two competing models. The sign of R indicates which
model is better. Since the value of R is estimated, we use the method proposed by Vuong, 1989 to
select the model. This function compares two models. The null hypothesis is that both classes of
distributions are equally far from the true distribution. If this is true, the log-likelihood ratio should
(asymptotically) have a Normal distribution with mean zero. The test statistic is the sample average
of the log-likelihood ratio, standardized by a consistent estimate of its standard deviation. If the
null hypothesis is false, and one class of distributions is closer to the "truth", the test statistic goes
to +/-infinity with probability 1, indicating the better-fitting class of distributions.

Usage

compare_distributions(d1, d2)

Arguments

d1 A distribution object
d2 A distribution object

Value

This function returns

test_statistic The test statistic.
p_one_sided A one-sided p-value, which is an upper limit on getting that small a log-likelihood
ratio if the first distribution, d1, is actually true.
p_two_sided A two-sided p-value, which is the probability of getting a log-likelihood ratio which
deviates that much from zero in either direction, if the two distributions are actually equally
good.

ratio A data frame with two columns. The first column is the x value and second column is the
difference in log-likelihoods.
Note

Code initially based on R code developed by Cosma Rohilla Shalizi (http://bactra.org/). Also see Appendix C in Clauset et al, 2009.

References


Examples

```r
# Example data
x = rpldis(100, xmin=2, alpha=3)

# Continuous power law
m1 = conpl$new(x)
m1$setXmin(estimate_xmin(m1))

# Exponential
m2 = conexp$new(x)
m2$setXmin(m1$getXmin())
est2 = estimate_pars(m2)
m2$setPars(est2$pars)

# Vuong's test
comp = compare_distributions(m1, m2)
plot(comp)
```

---

**conexp-class**

Heavy-tailed distributions

Description

The powerLaw package supports a number of distributions:

- **displ** Discrete power-law
- **dislnorm** Discrete log-normal
- **dispois** Discrete Poisson
- **disexp** Discrete Exponential
- **conpl** Continuous power-law
conlnorm Continuous log-normal
conexp Continuous exponential

Each object inherits the discrete_distribution or the ctn_distribution class.

Arguments

The object is typically created by passing data using the dat field. Each field has standard setters and getters.

Value

a reference object

Fields

Each distribution object has four fields. However, the object is typically created by passing data, to the dat field. Each field has standard setters and getters. See examples below

dat The data set.
xmin The lower threshold, xmin. Typically set after initialisation. For the continuous power-law, xmin >= 0 for the discrete distributions, xmin > 0
pars A parameter vector. Typically set after initialisation. Note the lognormal distribution has two parameters.
internal A list. This list differs between objects and shouldn’t be altered.

Copying objects

Distribution objects are reference classes. This means that when we copy objects, we need to use the copy method, i.e. obj$copy(). See the examples below for further details.

Examples

# Load data and create distribution object
#
# Load data

data(moby)

m = displ$new(moby)

# Xmin is initially the smallest x value
#
# Get Xmin and parameter

m$getXmin()
m$getPars()

# Set Xmin and parameter
#
# Set Xmin and parameter

m$setXmin(2)
m$setPars(2)
dist_all_cdf

The data cumulative distribution function

Description

This is generic function for distribution objects. This function calculates the data or empirical cdf.

The functions dist_data_all_cdf and dist_all_cdf are only available for discrete distributions. The main purpose is to optimise the bootstrap procedure, where generating a vector xmin:xmax is very quick. Also, when bootstrapping very large values can be generated.

Usage

dist_all_cdf(m, lower_tail = TRUE, xmax = 1e+05)
dist_data_cdf(m, lower_tail = TRUE, xmax = 1e+05)
dist_data_all_cdf(m, lower_tail = TRUE, xmax = 1e+05)

## S4 method for signature 'discrete_distribution'
dist_data_cdf(m, lower_tail = TRUE, xmax = 1e+05)

## S4 method for signature 'discrete_distribution'
dist_data_all_cdf(m, lower_tail = TRUE, xmax = 1e+05)

## S4 method for signature 'ctn_distribution'
dist_data_cdf(m, lower_tail = TRUE, xmax = 1e+05)
Arguments

- **m**: a distribution object.
- **lower_tail**: logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
- **xmax**: default 1e5. The maximum x value calculated when working out the CDF.

Note

This method does *not* alter the internal state of the distribution objects.

Examples

```r
# Load data and create distribution object
data(moby_sample)
# The data cdf
dist_all_cdf(m, lower_tail = TRUE, xmax = 1e5)
```

Description

This is a generic function for calculating the cumulative distribution function (cdf) of distribution objects. This is similar to base R’s `pnorm` for the normal distribution. The `dist_cdf` function calculates the cumulative probability distribution for the current parameters and xmin value.

Usage

```r
dist_cdf(m, q = NULL, lower_tail = FALSE)
```

```r
## S4 method for signature 'conexp'
dist_cdf(m, q = NULL, lower_tail = TRUE)
```

```r
## S4 method for signature 'conexp'
dist_all_cdf(m, lower_tail = TRUE, xmax = 1e5)
```

```r
## S4 method for signature 'conlnorm'
dist_cdf(m, q = NULL, lower_tail = TRUE)
```

```r
## S4 method for signature 'conlnorm'
dist_all_cdf(m, lower_tail = TRUE, xmax = 1e5)
```
dist_cdf

## S4 method for signature 'conpl'
```
dist_cdf(m, q = NULL, lower_tail = TRUE)
```

## S4 method for signature 'conpl'
```
dist_all_cdf(m, lower_tail = TRUE, xmax = 1e+05)
```

## S4 method for signature 'disexp'
```
dist_cdf(m, q = NULL, lower_tail = TRUE)
```

## S4 method for signature 'disexp'
```
dist_all_cdf(m, lower_tail = TRUE, xmax = 1e+05)
```

## S4 method for signature 'dislnorm'
```
dist_cdf(m, q = NULL, lower_tail = TRUE)
```

## S4 method for signature 'dislnorm'
```
dist_all_cdf(m, lower_tail = TRUE, xmax = 1e+05)
```

## S4 method for signature 'displ'
```
dist_cdf(m, q = NULL, lower_tail = TRUE)
```

## S4 method for signature 'displ'
```
dist_all_cdf(m, lower_tail = TRUE, xmax = 1e+05)
```

## S4 method for signature 'dispois'
```
dist_cdf(m, q = NULL, lower_tail = TRUE)
```

## S4 method for signature 'dispois'
```
dist_all_cdf(m, lower_tail = TRUE, xmax = 1e+05)
```

### Arguments

- **m**: a distribution object.
- **q**: a vector values where the function will be evaluated. If q is NULL (default), then the data values will be used.
- **lower_tail**: logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).
- **xmax**: default 1e5. The maximum x value calculated when working out the CDF.

### Note

This method does *not* alter the internal state of the distribution objects.

### Examples

```
# Load data and create distribution object
data(moby_sample)
```
m = displ$new(moby_sample)
m$setXmin(7); m$setPars(2)

# Calculate the CDF at a particular values#
dist_cdf(m, 10:15)

# Calculate the CDF at the data values#
dist_cdf(m)

---

dist_ll

The log-likelihood function

Description

This is generic function for distribution objects. This function calculates the log-likelihood for the current parameters and xmin value.

Usage

dist_ll(m)

## S4 method for signature 'conexp'
dist_ll(m)

## S4 method for signature 'conlnorm'
dist_ll(m)

## S4 method for signature 'conpl'
dist_ll(m)

## S4 method for signature 'disexp'
dist_ll(m)

## S4 method for signature 'dislnorm'
dist_ll(m)

## S4 method for signature 'displ'
dist_ll(m)

## S4 method for signature 'dispois'
dist_ll(m)

Arguments

m a distribution object.
**Value**

The log-likelihood

**Note**

This method does *not* alter the internal state of the distribution objects.

**See Also**

`dist_cdf`, `dist_pdf` and `dist_rand`

**Examples**

```r
# Load data and create distribution object
data(moby_sample)
m = disp$new(moby_sample)
m$setxmin(7); m$setPars(2)

dist_ll(m)
```

---

**dist_pdf**

*The probability density function (pdf)*

**Description**

This is a generic function for distribution objects. This function calculates the probability density function (pdf) for the current parameters and xmin value.

**Usage**

```r
dist_pdf(m, q = NULL, log = FALSE)
```

## S4 method for signature 'conexp'

dist_pdf(m, q = NULL, log = FALSE)

## S4 method for signature 'conlnorm'

dist_pdf(m, q = NULL, log = FALSE)

## S4 method for signature 'conpl'

dist_pdf(m, q = NULL, log = FALSE)

## S4 method for signature 'disexp'

dist_pdf(m, q = NULL, log = FALSE)
## dist_pdf

```r
## S4 method for signature 'dislnorm'
dist_pdf(m, q = NULL, log = FALSE)
```

```r
## S4 method for signature 'displ'
dist_pdf(m, q = NULL, log = FALSE)
```

```r
## S4 method for signature 'dispois'
dist_pdf(m, q = NULL, log = FALSE)
```

### Arguments

- `m` a distribution object.
- `q` a vector values where the function will be evaluated. If `q` is `NULL` (default), then the data values will be used.
- `log` default `FALSE`. If `TRUE`, probabilities are given as log(p).

### Value

The probability density (or mass) function

### Note

This method does *not* alter the internal state of the distribution objects.

### See Also

dist_cdf, dist_ll and dist_rand

### Examples

```r
# Create distribution object
m = displ$new()
m$setXmin(7); m$setPars(2)

dist_pdf(m, 7:10)
```
**Description**

This is a generic function for generating random numbers from the underlying distribution of the distribution reference objects. This function generates n random numbers using the parameters and xmin values found in the associated reference object.

**Usage**

```r
dist_rand(m, n)

## S4 method for signature 'conexp'
dist_rand(m, n = "numeric")

## S4 method for signature 'conlnorm'
dist_rand(m, n = "numeric")

## S4 method for signature 'conpl'
dist_rand(m, n = "numeric")

## S4 method for signature 'disexp'
dist_rand(m, n = "numeric")

## S4 method for signature 'dislnorm'
dist_rand(m, n = "numeric")

## S4 method for signature 'displ'
dist_rand(m, n = "numeric")

## S4 method for signature 'dispois'
dist_rand(m, n = "numeric")
```

**Arguments**

- `m` a distribution object.
- `n` number of observations to be generated.

**Value**

n random numbers

**Note**

This method does *not* alter the internal state of the distribution object.
See Also
dist_cdf, dist_pdf and dist_ll

Examples

```r
# Create distribution object
m = displ$new()
m$setXmin(7); m$setPars(2)

# Generate five random numbers
dist_rand(m, 5)
```

**dplcon**

The continuous powerlaw distribution

Density and distribution function of the continuous power-law distribution, with parameters xmin and alpha.

Description

The continuous powerlaw distribution

Density and distribution function of the continuous power-law distribution, with parameters xmin and alpha.

Usage

```r
dplcon(x, xmin, alpha, log = FALSE)
pplcon(q, xmin, alpha, lower.tail = TRUE)
rplcon(n, xmin, alpha)
```

Arguments

- `x, q` vector of quantiles. The discrete power-law distribution is defined for x > xmin
- `xmin` The lower bound of the power-law distribution. For the continuous power-law, xmin >= 0. for the discrete distribution, xmin > 0.
- `alpha` The scaling parameter: alpha > 1.
- `log` logical (default FALSE) if TRUE, log values are returned.
- `lower.tail` logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).
- `n` Number of observations. If length(n) > 1, the length is taken to be the number required.
**Value**

dplcon gives the density and pplcon gives the distribution function.

**Note**

The discrete random number generator is very inefficient

**Examples**

```r
xmin = 1; alpha = 1.5
x = seq(xmin, 10, length.out=1000)
plot(x, dplcon(x, xmin, alpha), type="l")
plot(x, pplcon(x, xmin, alpha), type="1", main="Distribution function")
n = 1000
con_rns = rplcon(n, xmin, alpha)
con_rns = sort(con_rns)
p = rep(1/n, n)
#Zipf's plot
plot(con_rns, rev(cumsum(p)), log="xy", type="l")
```

---

**dpldis**  
*Discrete powerlaw distribution*

**Description**

Density, distribution function and random number generation for the discrete power law distribution with parameters xmin and alpha.

**Usage**

```r
dpldis(x, xmin, alpha, log = FALSE)
ppldis(q, xmin, alpha, lower.tail = TRUE)
rpldis(n, xmin, alpha, discrete_max = 10000)
```

**Arguments**

- `x, q` vector of quantiles. The discrete power-law distribution is defined for x > xmin.
- `xmin` The lower bound of the power-law distribution. For the continuous power-law, xmin >= 0. for the discrete distribution, xmin > 0.
- `alpha` The scaling parameter: alpha > 1.
- `log` logical (default FALSE) if TRUE, log values are returned.
- `lower.tail` logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
- `n` Number of observations. If length(n) > 1, the length is taken to be the number required.
- `discrete_max` The value when we switch from the discrete random numbers to a CTN approximation.
Details

The Clausett, 2009 paper provides an algorithm for generating discrete random numbers. However, if this algorithm is implemented in R, it gives terrible performance. This is because the algorithm involves "growing vectors". Another problem is when alpha is close to 1, this can result in very large random number being generated (which means we need to calculate the discrete CDF for very large values).

The algorithm provided in this package generates true discrete random numbers up to 10,000 then switches to using continuous random numbers. This switching point can altered by changing the discrete_max argument.

In order to get a efficient power-law discrete random number generator, the algorithm needs to be implemented in C.

Value

dpldis returns the density, ppldis returns the distribution function and rpldis return random numbers.

Note

The naming of these functions mirrors standard R functions, i.e. dnorm. When alpha is close to one, generating random number can be very slow.

References


Examples

xmin = 1; alpha = 2
x = xmin:100

plot(x, dpldis(x, xmin, alpha), type="l")
plot(x, ppldis(x, xmin, alpha), type="l", main="Distribution function")
dpldis(1, xmin, alpha)

# Random number generation
n = 1e5
x1 = rpldis(n, xmin, alpha)
sum(x1==1)/n
x2 = rpldis(n, xmin, alpha, 0)
sum(x2==1)/n
**estimate_pars**  
*Estimates the distributions using mle.*

**Description**

`estimate_pars` estimates the distribution’s parameters using their maximum likelihood estimator. This estimate is conditional on the current `xmin` value.

**Usage**

```r
estimate_pars(m, pars = NULL)
```

**Arguments**

- `m`: A reference class object that contains the data.  
- `pars`: default `NULL`. A vector of parameters used to optimise over. Otherwise, for each value of `xmin`, the mle will be used, i.e. `estimate_pars(m)`. For small samples, the mle may be biased.

**Value**

returns list.

**Examples**

```r
data(moby_sample)  
m = displ$new(moby_sample)  
estimate_xmin(m)  
m$setXmin(7)  
estimate_pars(m)
```

---

**get_n**  
*Sample size*

**Description**

Returns the sample size of the data set contained within the distribution object.

**Usage**

```r
get_n(m)
```

**Arguments**

- `m`: a distribution object.
Examples

```
# Load data and create example object
data(moby_sample)
m = displ$new(moby_sample)

# get_n and length should return the same value
get_n(m)
length(moby_sample)
```

get_ntail  
Values greater than or equal to xmin

Description

Returns the number of data points greater than or equal to current value of xmin. In the Clauset et al, paper this is called 'ntail'.

Usage

```
get_ntail(m, prop = FALSE, lower = FALSE)
```

Arguments

- **m**: a distribution object.
- **prop**: default FALSE. Return the value as a proportion of the total sample size
- **lower**: default FALSE. If TRUE returns sample size - ntail

Examples

```
# Load data and create example object
data(moby_sample)
m = displ$new(moby_sample)
m$setXmin(7)

# Get ntail
get_ntail(m)
sum(moby_sample >= 7)
```
Generic plotting functions

Description

These are generic functions for distribution reference objects. Standard plotting functions, i.e. plot, points, and lines work with all distribution objects.

Usage

```r
## S4 method for signature 'distribution'
lines(x, cut = FALSE, draw = TRUE,
      length.out = 100, ...)

## S4 method for signature 'distribution,ANY'
plot(x, cut = FALSE, draw = TRUE, ...)

## S4 method for signature 'distribution'
points(x, cut = FALSE, draw = TRUE,
       length.out = 100, ...)
```

Arguments

- `x` a distribution reference object.
- `cut` logical (default FALSE) - Where should the plot begin. If cut=FALSE, then the plot will start at the minimum data value. Otherwise, the plot will start from `xmin`
- `draw` logical (default TRUE). Should the plot/lines/points function plot or return the data (in a data frame object).
- `length.out` numeric, default 100. How many points should the distribution be evaluated at. This argument is only for plotting the fitted lines.
- `...` Further arguments passed to the `lines` functions.

Note

This method does *not* alter the internal state of the distribution objects.
### Moby Dick word count

| Description | The frequency of occurrence of unique words in the novel Moby Dick by Herman Melville. The data set moby_sample is 2000 values sampled from the moby data set. |
| Format | A vector |

### Casualties in the American Indian Wars (1776 and 1890)

| Description | These data files contain the observed casualties in the American Indian Wars. The data sets native_american and us_american contain the casualties on the Native American and US American sides respectively. Each data set is a data frame, with two columns: the number of casualties and the conflict date. |
| Format | Data frame |
**plot.bs_xmin**  

Plot methods for bootstrap objects

**Description**  
A simple wrapper around the plot function to aid with visualising the bootstrap results. The values plotted are returned as an invisible object.

**Usage**  
```r  
## S3 method for class 'bs_xmin'  
plot(x, trim = 0.1, ...)  

## S3 method for class 'bs_p_xmin'  
plot(x, trim = 0.1, ...)  

## S3 method for class 'compare_distributions'  
plot(x, ...)  
```

**Arguments**  
- **x**  
  an object of class bs_xmin or bs_p_xmin
- **trim**  
  When plotting the cumulative means and standard deviation, the first trim percentage of values are not displayed. default trim=0.1
- **...**  
  graphics parameters to be passed to the plotting routines.

---

**population**  

City boundaries and the universality of scaling laws

**Description**  
This data set contains the population size of cities and towns in England. For further details on the algorithm used to determine city boundaries, see the referenced paper.

**Format**  
- vector

**Source**  
show, distribution-method

*Generic show method for distribution objects*

---

**Description**

The distribution objects have an internal structure that is used for caching purposes. Using the default `show` method gives the illusion of duplicate values. This show method aims to avoid this confusion.

**Usage**

```r
## S4 method for signature 'distribution'
show(object)
```

**Arguments**

- `object` A distribution object.

---

**swiss_prot**

*Word frequency in the Swiss-Prot database*

---

**Description**

This dataset contains all the words extracted from the Swiss-Prot version 9 data (with the resulting frequency for each word). Other datasets for other database versions can be obtained by contacting Michael Bell (http://homepages.cs.ncl.ac.uk/m.j.bell1/annotationQualityPaper.php)


**Format**

data frame

**Source**

# Index

*Topic **package**
  powerLaw-package, 2

bootstrap, 3
bootstrap_moby, 5
bootstrap_p (bootstrap), 3
bootstrap_p_moby (bootstrap_moby), 5

compare_distributions, 6
conexp (conexp-class), 7
conexp-class, 7
conlnorm (conexp-class), 7
conlnorm-class (conexp-class), 7
conpl (conexp-class), 7
conpl-class (conexp-class), 7
disexp (conexp-class), 7
disexp-class (conexp-class), 7
dislnorm (conexp-class), 7
dislnorm-class (conexp-class), 7
displ (conexp-class), 7
displ-class (conexp-class), 7
dispois (conexp-class), 7
dispois-class (conexp-class), 7
dist_all_cdf, 9
dist_all_cdfLconexo-method (dist_cdf), 10
dist_all_cdfLconexp-method (dist_cdf), 10
dist_all_cdfLconlnorm-method (dist_cdf), 10
dist_all_cdfLconpl-method (dist_cdf), 10
dist_all_cdfLdisexp-method (dist_cdf), 10
dist_all_cdfLdislnorm-method (dist_cdf), 10
dist_all_cdfLdispl-method (dist_cdf), 10
dist_all_cdfLdispois-method (dist_cdf), 10
dist_data_all_cdf (dist_all_cdf), 9
dist_data_all_cdfLdiscrete_distribution-method (dist_all_cdf), 9
dist_data_cdf (dist_all_cdf), 9
dist_data_cdf, ctn_distribution-method (dist_all_cdf), 9
dist_data_cdf, discrete_distribution-method (dist_all_cdf), 9
dist_ll, 12, 14, 16
dist_llLconexp-method (dist_ll), 12
dist_llLconlnorm-method (dist_ll), 12
dist_llLconpl-method (dist_ll), 12
dist_llLdisexp-method (dist_ll), 12
dist_llLdislnorm-method (dist_ll), 12
dist_llLdispl-method (dist_ll), 12
dist_llLdispois-method (dist_ll), 12
dist_pdf, 13, 13, 16
dist_pdfLconexp-method (dist_pdf), 13
dist_pdfLconlnorm-method (dist_pdf), 13
dist_pdfLconpl-method (dist_pdf), 13
dist_pdfLdisexp-method (dist_pdf), 13
dist_pdfLdislnorm-method (dist_pdf), 13
dist_pdfLdispl-method (dist_pdf), 13
dist_pdfLdispois-method (dist_pdf), 13
dist_rand, 13, 14, 15
dist_randLconexp-method (dist_rand), 15
dist_randLconlnorm-method (dist_rand), 15
dist_randLconpl-method (dist_rand), 15
dist_randLdisexp-method (dist_rand), 15
dist_randLdislnorm-method (dist_rand), 15

dist_cdf, 10, 13, 14, 16

dist_cdf, conexp-method (dist_cdf), 10
dist_cdf, conlnorm-method (dist_cdf), 10
dist_cdf, conpl-method (dist_cdf), 10
dist_cdf, disexp-method (dist_cdf), 10
dist_cdf, dislnorm-method (dist_cdf), 10
dist_cdf, displ-method (dist_cdf), 10
dist_cdf, dispois-method (dist_cdf), 10
dist_rand, displ-method (dist_rand), 15
dist_rand, dispois-method (dist_rand), 15
dplcon, 16
dpldis, 17

estimate_pars, 19
estimate_xmin (bootstrap), 3

get_KS_statistic (bootstrap), 3
get_n, 19
get_ntail, 20

lines, distribution-method, 21

moby, 22
moby_sample (moby), 22

native_american, 22
NativeAmerican (native_american), 22

plot, distribution, ANY-method
   (lines, distribution-method), 21
plot.bs_p_xmin (plot.bs_xmin), 23
plot.bs_xmin, 23
plot.compare_distributions
   (plot.bs_xmin), 23
points, distribution-method
   (lines, distribution-method), 21

Population (population), 23
population, 23
powerlaw (powerlaw-package), 2
powerlaw (powerlaw-package), 2
powerlaw-package, 2
powerlaw-package (powerlaw-package), 2
pplcon (dplcon), 16
ppldis (dpdis), 17

rplcon (dplcon), 16
rpldis (dpdis), 17

show, distribution-method, 24
Swiss_prot (swiss_prot), 24
swiss_prot, 24

us_american (native_american), 22
USAmerican (native_american), 22