The poweRlaw package

Colin S. Gillespie

Last updated: March 10, 2013

The `poweRlaw` package provides code to fit discrete and continuous power-law distributions. The fitting procedure follows the method detailed in Clauset *et al.*[1]. The scaling coefficient, α, is obtained by maximising the likelihood. The cut-off value, $x_{\min}$, is estimated by minimising the Kolmogorov-Smirnoff statistic.

Future versions of this package will allow other heavy tailed distributions to be fitted.

[1] A. Clauset, C.R. Shalizi, and M.E.J. Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009

1 *Installation*

The package is hosted on github.[2] The package can be installed using the `devtools` package:[3]

[2] The intention is eventually host this package on CRAN.

[3] If use Windows, you need to install the `Rtools` package first.

```
install.packages("devtools")
library(devtools)
install_github("poweRlaw", "csgillespie", subdir = "pkg")
```

Once installed, the package can be loaded ready for use with the standard `library` command

```
library(poweRlaw)
```

2 *Accessing documentation*

I have tried to ensure that the package and all associated functions and datasets are properly documented with runnable examples. The command

```
help(package = "poweRlaw")
```

will give a brief overview of the package and a complete list of all functions. The list of vignettes associated with the package can be obtained with

```
vignette(package = "poweRlaw")
```

At the time of writing, *this* vignette is the only one available, and can be accessed from the R command line with

```
vignette("poweRlaw", package = "poweRlaw")
```

Help on functions can be obtained using the usual R mechanisms. For example, help on the function `rpldis` can be obtained with

```
?rpldis
```

and the associated example can be run with

```
example(rpldis)
```

A list of demos and data sets associated with the package can be obtained with

```
demo(package = "poweRlaw")
data(package = "poweRlaw")
```

For example, the Moby dick data set can be load using

```
data(moby)
```

The package also contains the data set `moby_sample`. This data set is 2000 randomly sampled values from the larger `moby` data set.

After running this command, the vector `moby` will be accessible, and can be examined by typing

```
moby
```

at the R command prompt.

3 Example: Word frequency in Moby Dick

This example investigates the frequency of occurrence of unique words in the novel Moby Dick by Herman Melville[4]. The data can be downloaded from

http://tuvalu.santafe.edu/~aaronc/powerlaws/data.htm

or loaded directly

```
data(moby)
```

[4] A. Clauset, C.R. Shalizi, and M.E.J. Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009; and M.E.J. Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005

3.1 Fitting a discrete power-law

To fit a discrete power-law, we create a discrete power-law object, `displ`

```
m = displ$new(moby)
```

Initially the lower cut-off, $x_{\min}$ is set to the smallest x value and the scaling parameter, `alpha`, is set to `NULL`

```
m$getXmin()
## [1] 1
```

```
m$getPars()
## NULL
```

The distribution object also has standard setters

```
m$setXmin(5)
m$setPars(2)
```

For a given $x_{\min}$ value, we can estimate the corresponding α value using its maximum likelihood estimator (mle)

Instead of using the mle, we could instead do a parameter scan: `estimate_pars(m, pars=seq(2, 3, 0.1))`

```
estimate_pars(m)
## [1] 1.921
```

To estimate the lower bound, we minimise the distance between the data and the fitted model CDF, that is

$$D(x) = \max_{x \geq x_{\min}} |S(x) - P(x)|$$

where $S(x)$ is the data CDF and $P(x)$ is the theoretical CDF. The value $D(x)$ is known as the Kolmogorov-Smirnov statistic. Our estimate of $x_{\min}$ is then the value of x that minimises $D(x)$:

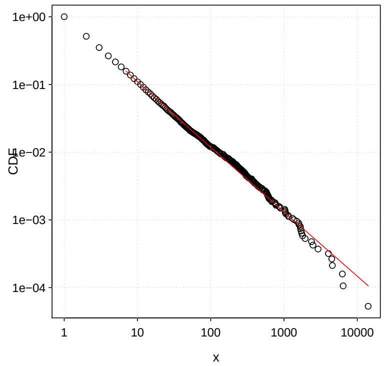

Figure 1: Plot of the data CDF for the Moby Dick data set. This corresponds to figure 6.1(a) in Clausett, 2009. Plot of data CDF with line of best fit.

```
(est = estimate_xmin(m))
## $KS
## [1] 0.009229
##
## $xmin
## [1] 7
##
## $pars
## [1] 1.95
##
## attr(,"class")
## [1] "ks_est"
```

We can then set parameters of power-law distribution to the "optimal" values

```
m$setXmin(est)
```

All distribution objects have generic plot methods:[5]

[5] Generic `lines` and `points` functions are also available.

```
## Plot the data (from xmin)
plot(m)
## Add in the fitted distribution
lines(m, col = 2)
```

When calling the `plot` and `lines` function, the data plotted is actually invisibly returned, i.e.

```
dd = plot(m)
head(dd, 3)
##   x      y
## 1 1 1.0000
## 2 2 0.5141
## 3 3 0.3505
```

Algorithm 1: Estimating the uncertainty in `xmin`

1: Calculate point estimates x_{min} and the scaling parameter α.
2: Set n_1 equal to the number of values below `xmin`.
3: Set $n_2 = n - n_1$.
4: **for** `i` in `1:B`:
5:     Simulate n_1 values from a discrete uniform distribution: $U(1, x_{\text{min}})$ and n_2 values from a discrete power-law (with parameter α).
6:     Estimate x_{min} and α using the Kolmogorov-Smirnoff statistic.
7: **end for**

3.2 *Quantifying x_{min} uncertainty*

Clausett, *el al*, 2009 recommend a bootstrap procedure to estimate the uncertainty in x_{min}. Essentially, an estimate of parameter uncertainty is obtained by generating multiple data sets (with parameters x_{min} and α) and then "re-inferring" the model parameters. The algorithm is detailed in Algorithm 1.[6]

[6] Algorithm 1 can easily be extended for other distributions.

When α is close to one, this algorithm can be particularly time consuming to run, for two reasons:

1. When generating random numbers from the discrete power-law distribution, extreme values are highly possible, i.e. values greater than 10^8. Hence, when generating the random numbers, all numbers larger than 10^5 are generated using a continuous approximation.

2. To calculate the Komologorov-Smirnov statistic, we need explore the state space. It is computationally infeasible to explore the entire state space when $\max(x) >> 10^5$. So to this algorithm feasible, we explore two state space. The first,

$$x_{\text{min}}, x_{\text{min}} + 1, x_{\text{min}} + 2, \ldots, 10^5$$

   and combine it with an additional 10^5 values from

$$10^5, \ldots, \max(x)$$

The bootstrapping procedure, steps 4 – 7, can be run in parallel. To estimate the uncertainty with the `moby` data set, we use

```
## This takes a while
bs = bootstrap_xmin(m, no_of_sims = 1000, threads = 1)
```

The object returned from the bootstrap procedure contains three elements

- A p-value .. `bs$p`
- The original goodness of fit statistic - `bs$gof`
- The result of the bootstrap procedure - a data frame with three columns.
- The average time (in seconds) for a single bootstrap realisation.

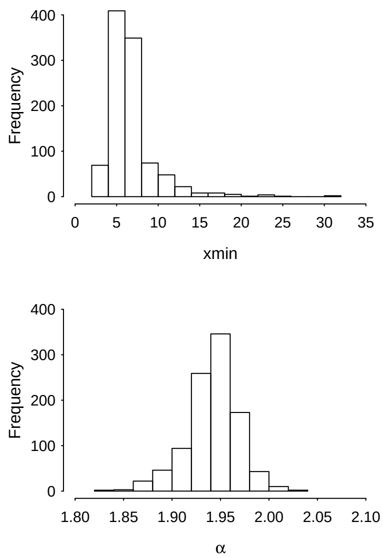

Figure 2: Histograms of the bootstrap results.

The results of the bootstrap are best investigated with histograms

```
hist(bs$bootstraps[, 2])
hist(bs$bootstraps[, 3])
```

and a bivariate scatter plot

```
plot(bs$bootstraps[, 2], bs$bootstraps[, 3])
```

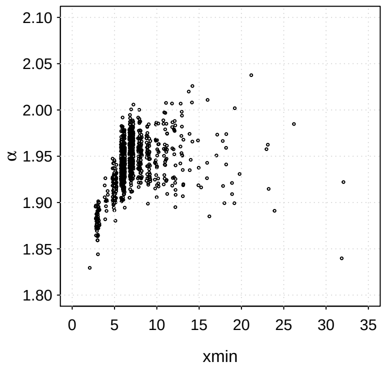

Figure 3: Bivariate scatter plot of the bootstrap results. The values of $x_{\min}$ and α are obviously strongly correlated.

4 *Distribution objects*

During the Moby Dick example, we created a `displ` object

```
m = displ$new(moby)
```

The object `m` has class `displ` and inherits the general `distribution` class. A list of available distributions is given in table 1.

Distribution	Object name	# Parameters
Discrete Power-law	`displ`	1
CTN Power-law	`conpl`	1

Table 1: Available distributions in the power-law package. These objects are all reference classes.

All distribution objects list in table 1 are reference classes. The key point, is that unlike S4 classes, reference classes have a mutable set. Each distribution object has four fields:

See `?setRefClass` for further details on references classes.

- `datatype`: This will be set to *discrete* or *continuous*.
- `dat`: a copy of the data.
- `xmin`: the lower cut-off $x_{\min}$.
- `pars`: a vector of parameter values.
- `internal`: a list of values use in different numerical procedures. This will differ between distribution objects.

By using the mutable state, we have efficient caching of data structures that can be reused. For example, the mle of discrete power-laws uses the statistic:

$$\sum_{i=x_{\min}}^{n} \log(x_i)$$

This value is calculated once for all values of $x_{\min}$, then iterated over when estimating $x_{\min}$.

All distribution objects have a number of methods available. A list of methods is given in table 2. See the associated help files for further details.

Method Name	Description
`dist_cdf`	Cumulative density function
`dist_pdf`	Probability density function
`dist_rand`	Random numbers generator
`dist_data_cdf`	Data CDF
`dist_ll`	Log-likelihood
`estimate_xmin`	Estimate the cut-off point and parameter value
`estimate_pars`	Estimate of the parameters (conditional on the current $x_{\min}$ value)
`bootstrap_xmin`	Bootstrap procedure

Table 2: A list of functions for `distribution` functions. These objects do not change the object states. However, they may not be thread safe.

5 *Loading data*

Typically, data is stored in a csv or text file. To use this data, we load it in the usual way[7]

```
blackouts = read.table("blackouts.txt")
```

Distribution objects take vectors as inputs, so

```
m_bl = conpl$new(blackouts$V1)
```

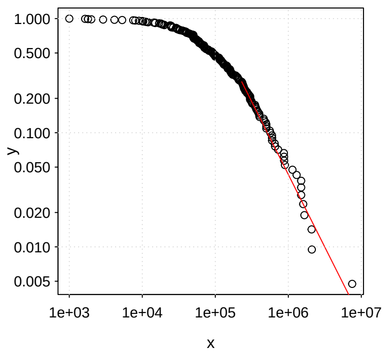

Figure 4: CDF plot of the blackout dataset with line of best fit. Since the minimum value of x is large, we fit a continuous power-law as this is more it efficient.

[7] The blackouts data set can be obtained from Clauset's website: http://goo.gl/BsqnP .

6 *Comparison with the* `plfit` *script*

6.1 *The discrete case*

Other implementations of estimating the lower bound can be found at

http://tuvalu.santafe.edu/~aaronc/powerlaws/

In particular, the script for estimating $x_{\min}$ can be loaded using

```
source("http://tuvalu.santafe.edu/~aaronc/powerlaws/plfit.r")
```

The results are directly comparable to the poweRlaw package. For example, if we look consider the Moby Dick data set again:

```
plfit(moby)
## $xmin
## [1] 7
##
## $alpha
## [1] 1.95
##
## $D
## [1] 0.009289
```

Notice that the results are slightly different. This is because the `plfit` by default does a parameter scan over the range

$$1.50, 1.51, 1.52, \ldots, 2.49, 2.50$$

To exactly replicate the results, we could use

```
estimate_xmin(m, pars = seq(1.5, 2.5, 0.01))
```

6.2 *The continuous case*

The `plfit` script also fits continuous power-laws. Again the results are comparable. First we generate one thousand random numbers from the continuous power-law with parameters $\alpha = 2.5$ and $x_{\min} = 10.0$

```
r = rplcon(1000, 10, 2.5)
```

The `plfit` automatically detects if the data is continuous

```
plfit(r)
## $xmin
## [1] 10.93
##
## $alpha
## [1] 2.487
##
## $D
## [1] 0.02223
```

Fitting with the `poweRlaw` package gives the same values

```
m_r = conpl$new(r)
(est = estimate_xmin(m_r))
## $KS
## [1] 0.02223
##
## $xmin
## [1] 10.93
##
## $pars
## [1] 2.487
##
## attr(,"class")
## [1] "ks_est"
```

Of course, using the `poweRlaw` package, we can easily plot the data

```
m_r$setXmin(est)
plot(m_r)
lines(m_r, col = 2)
```

to get figure 5.

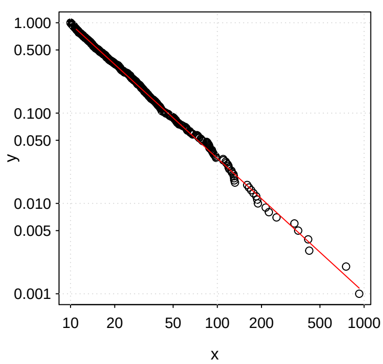

Figure 5: CDF plot of one thousand random numbers generated from a power-law with parameters $\alpha = 2.5$ and $x_{\min} = 10$. The line of best fit is also shown.

References

[1] A. Clauset, C.R. Shalizi, and M.E.J. Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[2] M.E.J. Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

Package and R version

Package	Version
parallel	2.15.3
poweRlaw	0.16.0
VGAM	0.9-0

Table 3: A list of packages and versions used.

```
version
##                _
## platform       x86_64-pc-linux-gnu
## arch           x86_64
## os             linux-gnu
## system         x86_64, linux-gnu
## status
## major          2
## minor          15.3
## year           2013
## month          03
## day            01
## svn rev        62090
## language       R
## version.string R version 2.15.3 (2013-03-01)
## nickname       Security Blanket
```