maxlogL: A general computational procedure for Maximum Likelihood estimation in R

Jaime Mosquera Gutiérrez
Universidad Nacional de Colombia

Freddy Hernández
Universidad Nacional de Colombia

Abstract

Maximum likelihood (ML) method is preferred among others because it produces consistent and efficient estimators. However, likelihood optimization processes frequently involve unwieldy mathematical expressions and it is necessary in some cases to implement distributions and constantly build (log-)likelihood functions in computing languages in order to get numerical solutions. We present `maxlogL`, a function contained in the **EstimationTools** package for maximum likelihood parameter estimation of any probability function loaded in R with no need of special structures, given a data set. Finally, we present two application examples with real data.

Keywords: Maximum likelihood estimation, parameter estimation, R, **EstimationTools**.

1. Introduction

Parameter estimation for probability density functions or probability mass functions is a central problem in statistical analysis and applied sciences because it allows to build predictive models and make inferences. Traditionally this problem has been tackled by means of likelihood maximization, as it was introduced by Fisher (1912). The method consists on performing an optimization through first derivative of the log-likelihood function and solve the outcoming system of equations. Despite its validity for any probability distribution, there exist a vast variety of them with cumbersome derivatives which produce non-linear systems of equations, therefore it is necessary to implement numerical methods and develop computational algorithms to find a solution.

R (R Core Team 2019) is a free language developed for statistical computing and equipped with unconstrained and box-constrained general-purpose optimization tools in **base** package: Fox, Hall, and Schryer (1978) developed the function `nlminb` for optimization using PORT (portable Fortran programs for numerical computation) routines; Nash (1979) implemented `optim`, a function that performs optimization based on three algorithms: (1) Nelder and Mead (1965), (2) quasi-Newton (`BFGS`) and (3) conjugate-gradient (`CG`). Either the former or the latter is implemented, users must take a distribution included in any package or create their own function otherwise and then write the likelihood.

On the other hand, R posses an extensive number of libraries (add-ons) in order to enhance its capabilities, e.g **gamlss** package (Stasinopoulos and Rigby 2007) for fitting generalized additive models for location, scale and shape. It is possible to carry out parameter estimation with an empty regression model of any distribution implemented as a `gamlss.family` structure.

Visit Stasinopoulos, Rigby, Heller, Voudouris, and De Bastiani (2017) for more details.

In this paper, we introduce the function `maxlogL`, which is capable of applying maximum likelihood estimation based on optimization through `optim` or `nlminb` only with the density/mass function defined as usual in R. The user could define its own distribution or use whichever existing in any package. The remainder of the article defines the maximization problem mathematically and computationally. Then, we present a simulation study to evaluate the performance of `maxlogL` with data generated from normal, ZIP and user-defined distributions. Finally, we give an application with a real data set and present some conclusions.

2. Maximum Likelihood estimation

Let be $\boldsymbol{y}^\top = (y_1, y_2, ..., y_n)$ a random sample with n observations drawn from a population with distribution $f(\cdot|\boldsymbol{\theta})$, with $\boldsymbol{\theta}$ a vector of parameters. The likelihood function of $\boldsymbol{\theta}$ is

$$L(\boldsymbol{\theta}|\boldsymbol{y}) = \prod_{i=1}^{n} f(y_i|\boldsymbol{\theta}). \tag{1}$$

The method of ML finds the parameter values that makes data more probable. It is achieved by computing a vector $\hat{\boldsymbol{\theta}}$ such that

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in\Theta}{\arg\max}\; L(\boldsymbol{\theta}|\boldsymbol{y}). \tag{2}$$

It is usual to perform maximization of log-likelihood function, i.e. $l(\boldsymbol{\theta}|\boldsymbol{y}) = \log L(\boldsymbol{\theta}|\boldsymbol{y})$. The variance-covariance matrix of ML estimators is given by

$$Var(\hat{\boldsymbol{\theta}}) = \mathcal{J}^{-1}(\hat{\boldsymbol{\theta}}) = C(\hat{\boldsymbol{\theta}}), \tag{3}$$

where $\mathcal{J}(\hat{\boldsymbol{\theta}})$ is the observed Fisher Information Matrix

$$\mathcal{J}(\hat{\boldsymbol{\theta}}) = \frac{\partial^2}{\partial\boldsymbol{\theta}^2}\log(f(y_i|\boldsymbol{\theta})). \tag{4}$$

The standard errors can be calculated as the square root of the diagonal elements of matrix C (Pawitan 2013)

$$S.E(\hat{\boldsymbol{\theta}}) = \sqrt{C_{jj}(\hat{\boldsymbol{\theta}})}. \tag{5}$$

The R function presented here calculates $l(\boldsymbol{\theta}|\boldsymbol{y})$ computationally, and computes standard errors from Hessian matrix.

3. Basic usage and features

Our `maxlogL` function is a S3 object of class `maxlogL`. It is included in **EstimationTools**, a package available from the Comprehensive R Archive Network (CRAN) `https://cran.r-project.org/package=EstimationTools`. It can be downloaded and loaded in global environment typing the following instructions in the console:

```
R> install.packages("EstimationTools")
R> library(EstimationTools)
```

With `maxlogL` we provide a flexible implementation of ML estimation. It cab be executed stating its most important arguments

```
R> maxlogL(x, dist, optimizer, lower = NULL, upper = NULL)
```

where the argument `x` is a vector with data to be fitted, `dist` corresponds to the probability density/mass function of the working distribution, whereas `upper` and `lower` are limits used when user selects box-constrained algorithms. `maxlogL` is a wrapper function specifically developed for ML estimation, which allows to implement any of `optim` algorithms for optimization or `nlminb` routine for unconstrained or box-constrained optimization through the argument `optimizer`.

EstimationTools package provides a `summary` method for class `maxlogL`, which displays AIC (Akaike Information Criterion), BIC (Bayesian Information Criterion), ML estimates and its standard error. The method also reports the optimization routine selected by the user and the method used for computation of standard error. There are three methods available: `hessian` function from **numDeriv** package, calculation with `optim` (setting the argument `hessian = TRUE`) and bootstrap calculation, with `boot` function of **boot** package (Davison and Hinkley 1997; Canty and Ripley 2017).

Hence, for non-censorship fitting the user must pass a vector with data and specify a probability distribution function available in R. For example, fitting a sample generated from a normal distribution, $Z \sim \text{NO}(\mu = 10, \sigma^2 = 1)$, could be done with the next command lines:

```
R> set.seed(1000)
R> z <- rnorm(n = 1000, mean = 10, sd = 1)
R> fit1 <- maxlogL(x = z, dist = 'dnorm', start=c(2, 3),
+                  lower=c(-15, 0), upper=c(15, 10))
R> summary(fit1)
```

```
---------------------------------------------------------------
Optimization routine: nlminb
Standard Error calculation: Hessian from optim
---------------------------------------------------------------
       AIC      BIC
  2804.033 2813.849
---------------------------------------------------------------
     Estimate  Std. Error Z value Pr(>|z|)
mean   9.98752    0.03103  321.87   <2e-16 ***
sd     0.98126    0.02194   44.72   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
---------------------------------------------------------------
Note: p-values valid under asymptotic normality of estimators
---
```

`maxlogL` also reports the optimization routine selected by the user and the method used for computation of standard error (see the `summary` output above). There are three options available:

- If `StdE_Method = optim`, the Hessian matrix is computed with `optim` (with option 'hessian = TRUE' under the hood).
- If the previous implementation fails or if the user chooses `StdE_Method = numDeriv`, it is calculated with `hessian` function from **numDeriv** package.

Hence, the standard errors are computed with the inverse Hessian (recall section **??**). Additionally, then standard errors and estimates can be computed by bootstrapping, even if the Hessian computation does not fail. It can be achieved with the function `bootstrap_maxlogL`, which consist of a implementation of `boot` function of **boot** package (Davison and Hinkley 1997; Canty and Ripley 2017).

The `link` argument is a list with entries `fun` and `over`, which specify the link functions applied and the name of linked parameters in probability function implemented in R respectively. The estimation performed above can be carried out applying logarithm link function to σ, avoiding problems of estimation in the boundary of parametric space. The usage is illustrated in the following code snippet:

```
R> fit2 <- maxlogL(x = z, dist = 'dnorm',
+                  link = list(over = "sd", fun = "log_link"))
R> summary(fit2)
```

```
_______________________________________________________________
Optimization routine: nlminb
Standard Error calculation: Hessian from optim
_______________________________________________________________
       AIC      BIC
  2804.033 2813.849
_______________________________________________________________
     Estimate  Std. Error Z value Pr(>|z|)
mean   9.98752    0.03103  321.87   <2e-16 ***
sd     0.98126    0.02194   44.72   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
_______________________________________________________________
Note: p-values valid under asymptotic normality of estimators
---
```

The user can apply link functions to more than one parameter of the distribution:

```
R> fit3 <- maxlogL(x = z, dist = 'dnorm',
+                  link = list(over = c("mean", "sd"),
+                              fun = c("log_link", "log_link")))
R> summary(fit3)
```

```
--------------------------------------------------------------
Optimization routine: nlminb
Standard Error calculation: Hessian from optim

--------------------------------------------------------------
       AIC      BIC
  2804.033 2813.849

--------------------------------------------------------------
     Estimate  Std. Error Z value Pr(>|z|)
mean   9.98752    0.03103  321.87   <2e-16 ***
sd     0.98126    0.02194   44.72   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

--------------------------------------------------------------
Note: p-values valid under asymptotic normality of estimators
---
```

Other link functions available are logit and negative inverse, which must be specified as `"logit_link"` and `"NegInv_link"`. On the other hand, `maxlogL` allows to define fixed known parameters, e.g., the sample size n in estimation of success proportion in a binomial distribution $N \sim \mathrm{BI}(p = 0.3, n = 10)$. This parameters can be specified with `fixed` argument, which is a list that stores the fixed parameters value specified by their names

```
R> set.seed(100)
R> N <- rbinom(n = 100, size = 10, prob = 0.3)
R> phat <- maxlogL(x = N, dist = 'dbinom', fixed = list(size = 10),
+                 link = list(over = "prob", fun = "logit_link"))
R> summary(phat)
```

```
--------------------------------------------------------------
Optimization routine: nlminb
Standard Error calculation: Hessian from optim

--------------------------------------------------------------
       AIC      BIC
  334.9805 334.9805

--------------------------------------------------------------
     Estimate  Std. Error Z value Pr(>|z|)
prob   0.31200    0.01465    21.3   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

--------------------------------------------------------------
Note: p-values valid under asymptotic normality of estimators
---
```

As can be seen, the procedure applies the inverse of the link function to return the parameter to the original scale.

4. Illustrative examples

In the following examples we replicate maximum likelihood method with `maxlogL` in two applications: fitting of a power Lindley distribution to model tensile strength of carbon fibers and parameter estimation in two stage hierarchical model of retention proportions in memory tests.

4.1. Tensile strength data: power Lindley distribution

1.312	1.314	1.479	1.552	1.700	1.803	1.861	1.865	1.944	1.958	1.966
1.997	2.006	2.027	2.055	2.063	2.098	2.14	2.179	2.224	2.240	2.253
2.270	2.272	2.274	2.301	2.301	2.359	2.382	2.382	2.426	2.434	2.435
2.478	2.490	2.511	2.514	2.535	2.554	2.566	2.57	2.586	2.629	2.633
2.642	2.648	2.684	2.697	2.726	2.770	2.773	2.800	2.809	2.818	2.821
2.848	2.88	2.954	3.012	3.067	3.084	3.090	3.096	3.128	3.233	3.433
3.585	3.585									

Table 1: Tensile strength of 69 fibers (Devendra and Rangaswamy 2013).

Data presented in Table 1 correspond to the tensile strength T (in GPa) of 69 specimens of carbon fiber tested under tension at gauge lengths of 20 mm.

Ghitany, Al-Mutairi, Balakrishnan, and Al-Enezi (2013) fitted their power Lindley (PL) distribution:

$$f(u|\mu,\sigma) = \frac{\mu\sigma^2}{\sigma+1}(1+u^{\mu})u^{\mu-1}e^{-\sigma u^{\mu}}, \quad u>0,\ \mu,\sigma>0. \tag{6}$$

We implemented this density function in the R function `dPL` displayed below:

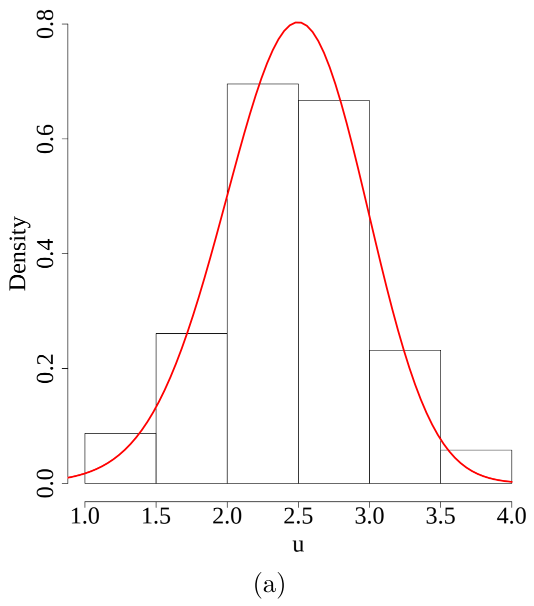
```
R> dPL <- function(x, mu, sigma, log=FALSE){
+    if (any(x < 0))
+      stop(paste("x must be positive", "\n", ""))
+    if (any(mu <= 0))
+      stop(paste("mu must be positive", "\n", ""))
+    if (any(sigma <= 0))
+      stop(paste("sigma must be positive", "\n", ""))
+
+    loglik <- log(mu) + 2*log(sigma) - log(sigma+1) +
+      log(1+(x^mu)) + (mu-1)*log(x) - sigma*(x^mu)
+
+    if (log == FALSE)
+      density <- exp(loglik)
+    else density <- loglik
+    return(density)
+  }
```

Then, we estimate parameters with `maxlogL` taking the vector of strengths from data set `fibers`, as follows:

```
R> # Fitting of tensile strenght data
R> st <- Fibers$Strenght
R> theta <- maxlogL(x = st, dist = "dPL",
+                   link = list(over = c("mu", "sigma"),
+                               fun = c("log_link", "log_link")))
R> summary(theta)
```

```
---------------------------------------------------------------
Optimization routine: nlminb
Standard Error calculation: Hessian from optim
---------------------------------------------------------------
      AIC      BIC
  102.119 106.5872
---------------------------------------------------------------
      Estimate  Std. Error Z value Pr(>|z|)
mu      3.86778    0.31371  12.329  < 2e-16 ***
sigma   0.04967    0.01599   3.107  0.00189 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
---------------------------------------------------------------
Note: p-values valid under asymptotic normality of estimators
---
```

Estimations are $\hat{\mu} = 3.8678$ and $\hat{\sigma} = 0.0497$. Essentially, we get the same values computed by Ghitany *et al.* (2013). In Figure 1 we showed the performance of parameter estimation plotting corresponding density along with the histogram in the left panel and estimated survival function along with Kaplan-Meier estimator in the right panel.



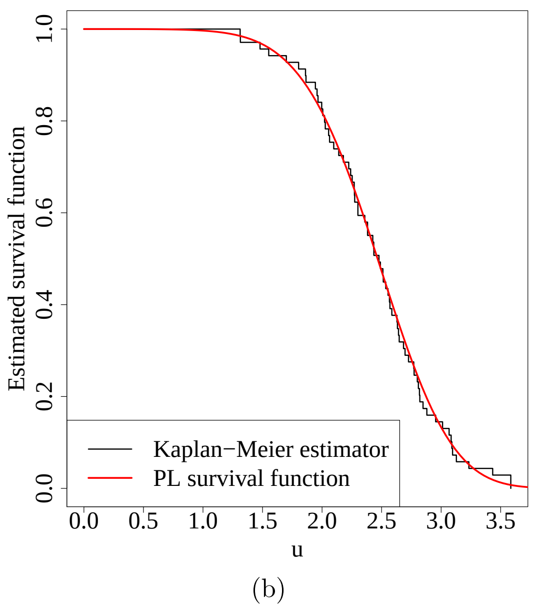


Figure 1: Fitting of tensile strength data: (a) estimated density and histogram; (b) estimated power Lindley survival function and Kaplan-Meier estimator.

4.2. Forgetting data: hierarchical binomial distribution

`maxlogL` is capable of fit hierarchical models with the proper function definition of input variable. To illustrate the estimation, we replicated forgetting data example presented in Myung (2003), which used data from Murdock, Bennet B. (1961).

The retention function is a probability function that models the proportion of correct recall at time t_i in each trial in memory tests. Myung (2003) studied the following two models in their application example:

$$\begin{aligned} \text{power model: } p(a, b, t) &= at^{-b}, \\ \text{exponential model: } p(a, b, t) &= ae^{-bt}, \quad a, b > 0. \end{aligned} \tag{7}$$

$m = 100$ (number of trials)						
Retention Interval (sec.)	1	3	6	9	12	18
Observed proportion	0.94	0.77	0.40	0.26	0.24	0.16

Table 2: Observed proportion of recalls at each time.

Each observation in the data set corresponds to a proportion obtained as the quotient of correct responses (x_i) and the total number of independent trials (replications of each memory test, represented by m). This kind of experiment can be modelled with a binomial distribution:

$$f(x_i|a, b, m) = \frac{m!}{(m - x_i)!\, x_i!} p(a, b, t_i) \left[1 - p(a, b, t_i)\right]^{m - x_i} \tag{8}$$

The usefulness of each retention equation is given by their goodness of fit.

Power model implementation

The hierarchical model with power retention function is implemented in an R function as follows:

```
R> # Power model implementation
R> power_logL <- function(x, a, b, log = FALSE){
+    p <- a * x[,1]^(-b)
+    f <- dbinom(x = x[,2], size = m, prob = p)
+    if (log == TRUE)
+      density <- log(f) else density <- f
+    return(density)
+  }
```

Conditional density in equation (8) depends on x_i, but proportion of successes depends on t, as equation (7) shows. For this reason, the input argument `x` must be a $n \times 2$ matrix, where n is the sample size. In forgetting data, $n = 6$. Then, the estimation is performed as usual with `maxlogL`. Note line six in the following chunk of code, which corresponds to the matrix definition of the input data aforementioned:

```
R> # Power model estimation
R> m <- 100 # Independent trials
R> t <- c(1,3,6,9,12,18) # time intervals
R> p.ob <- c(0.94,0.77,0.40,0.26,0.24,0.16) # Observed proportion
R> x <- p.ob*m # Correct responses
R> x <- as.integer(x)
R> Xi <- matrix(c(t,x), ncol=2, nrow=6)
R> retention.pwr <- maxlogL(x = Xi, dist = "power_logL", lower = c(0.01,0.01),
+                             upper = c(1,1), start = c(0.1,0.1))
R> summary(retention.pwr)

---------------------------------------------------------------
Optimization routine: nlminb
Standard Error calculation: Hessian from optim
---------------------------------------------------------------
      AIC    BIC
  57.4522 58.422
---------------------------------------------------------------
  Estimate  Std. Error Z value Pr(>|z|)
a   0.95312    0.01860   51.25   <2e-16 ***
b   0.49793    0.03236   15.38   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
---------------------------------------------------------------
Note: p-values valid under asymptotic normality of estimators
---
```

In this application, convergence was achieved by solving a box-constrained likelihood optimization, whose boundaries were specified in arguments `lower` and `upper`. Furthermore, tuning initial values was necessary with argument `start`. Computed values are $\hat{\boldsymbol{\theta}} = (\hat{a}_{\text{pwr}}, \hat{b}_{\text{pwr}}) = (0.953, 0.498)$, which are the same estimates of Myung (2003). Power fitting is illustrated in Figure 2.

Exponential model implementation

Similarly as before, exponential retention function is implemented and input data is defined as a matrix, in this fashion:

```
R> # Exponential model implementation
R> exp_logL <- function(x, a, b, log = FALSE){
+    p <- a * exp(-x[,1]*b)
+    f <- dbinom(x = x[,2], size = m, prob = p)
+    if (log == TRUE)
+      density <- log(f) else density <- f
+    return(density)
+  }
R> # Exponential model estimation
R> m <- 100 # Independent trials
R> t <- c(1,3,6,9,12,18) # time intervals
R> p.ob <- c(0.94,0.77,0.40,0.26,0.24,0.16) # Observed proportion
R> x <- p.ob*m # Correct responses
R> x <- as.integer(x)
R> Xi <- matrix(c(t,x), ncol=2, nrow=6)
R> retention.exp <- maxlogL(x = Xi, dist = 'exp_logL', lower = c(0.1,0.1),
+                          upper = c(2,2), start = c(0.1,0.2))
R> summary(retention.exp)

_______________________________________________________________
Optimization routine: nlminb
Standard Error calculation: Hessian from optim
_______________________________________________________________
      AIC     BIC
  41.3329 42.3027
_______________________________________________________________
  Estimate  Std. Error Z value Pr(>|z|)
a  1.070112   0.031342   34.14   <2e-16 ***
b  0.130826   0.009252   14.14   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
_______________________________________________________________
Note: p-values valid under asymptotic normality of estimators
---
```

Again, we obtain the same estimates of Myung (2003): $\hat{\boldsymbol{\theta}} = (\hat{a}_{\text{exp}}, \hat{b}_{\text{exp}}) = (1.070, 0.131)$.

Exponential fitting is shown in Figure 2. According to the Akaike information criterion, the exponential model has better fitness ($AIC_{\text{exp}} = 41.33$ against $AIC_{\text{pwr}} = 57.45$).

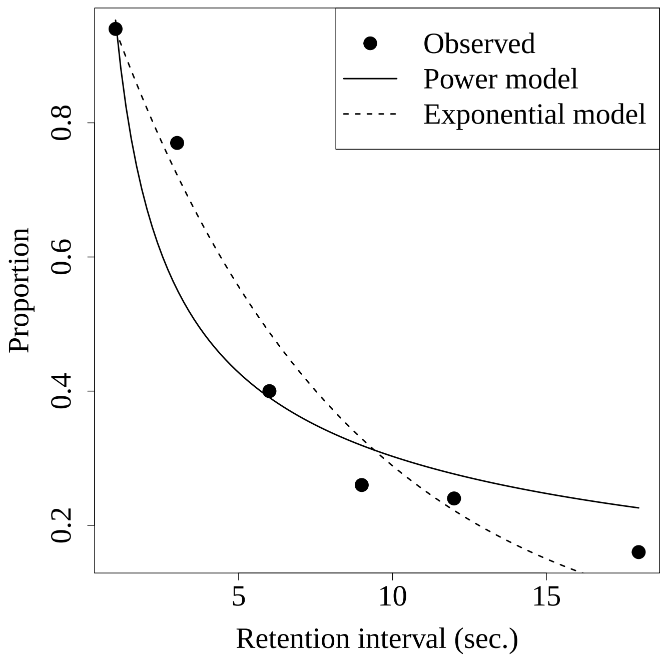


Figure 2: Observed proportion of recalls and models fitted.

5. Conclusions

We have implemented classic estimation via maximization of log-likelihood function through basic optimization routines in R such as `optim`, `nlminb` and `DEoptim`. With `maxlogL`, we enable researchers, developers and users in general to compute ML estimators of any distribution in a fast and reliable way. With our `summary` method, it is possible to calculate standard error of estimates through Hessian matrix or bootstrap algorithm. In some cases, it is possible to implement estimation in hierarchical models, with appropriate tuning of initial values.

In future revisions, we could implement evolutionary algorithms to perform estimation in distributions with regularity issues (Haupt and Haupt 2003). Furthermore, our routine could be took to develop further work on log-likelihood estimation, such as developing new parametric regression models.

References

Canty A, Ripley BD (2017). *boot: Bootstrap R (S-Plus) Functions.*

Davison AC, Hinkley DV (1997). *Bootstrap Methods and Their Applications.* Cambridge University Press, Cambridge. URL `http://statwww.epfl.ch/davison/BMA/`.

Devendra K, Rangaswamy T (2013). "Strength Characterization of E-glass Fiber Reinforced Epoxy Composites with Filler Materials." *Journal of Minerals and Materials Characterization and Engineering*, **01**(06), 353–357. ISSN 2327-4077. `doi:10.4236/jmmce.2013.16054`. URL `http://www.scirp.org/journal/doi.aspx?DOI=10.4236/jmmce.2013.16054`.

Fisher R (1912). "On an Absolute Criterion for Fitting Frequency Curves." *Messenger of Mathematics*, **41**(1), 155–160. ISSN 08834237. `doi:10.2307/2246266`.

Fox PA, Hall AP, Schryer NL (1978). "The PORT Mathematical Subroutine Library." *ACM Transactions on Mathematical Software*, **4**(2), 104–126. ISSN 00983500. `doi:10.1145/355780.355783`. URL `https://dl.acm.org/doi/10.1145/355780.355783`.

Ghitany ME, Al-Mutairi DK, Balakrishnan N, Al-Enezi LJ (2013). "Power Lindley distribution and associated inference." *Computational Statistics and Data Analysis*, **64**, 20–33. ISSN 01679473. `doi:10.1016/j.csda.2013.02.026`. URL `http://dx.doi.org/10.1016/j.csda.2013.02.026`.

Haupt RL, Haupt SE (2003). *Practical Genetic Algorithms.* John Wiley & Sons, Inc., Hoboken, NJ, USA. ISBN 0471455652. `doi:10.1002/0471671746`. URL `http://doi.wiley.com/10.1002/0471671746`.

Murdock, Bennet B J (1961). "The retention of individual items." *Journal of Experimental Psychology*, **62**(6), 618–625. ISSN 0022-1015. `doi:10.1037/h0043657`. URL `http://content.apa.org/journals/xge/62/6/618`.

Myung IJ (2003). "Tutorial on maximum likelihood estimation." *Journal of Mathematical Psychology*, **47**(1), 90–100. ISSN 00222496. `doi:10.1016/S0022-2496(02)00028-7`. URL `https://linkinghub.elsevier.com/retrieve/pii/S0022249602000287`.

Nash JC (1979). *Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation.* 2nd editio edition. Adam Hilger, Bristol.

Nelder JA, Mead R (1965). "A Simplex Method for Function Minimization." *The Computer Journal*, **7**(4), 308–313. ISSN 0010-4620. `doi:10.1093/comjnl/7.4.308`. URL `https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/7.4.308`.

Pawitan Y (2013). *In all likelihood: statistical modelling and inference using likelihood.* Oxford University Press. ISBN 978-0199671229.

R Core Team (2019). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL `https://www.r-project.org/`.

Stasinopoulos DM, Rigby RA (2007). "Generalized Additive Models for Location Scale and Shape ({GAMLSS}) in \proglang{R}." *Journal of Statistical Software*, **23**(7), 1–46. `doi:10.18637/jss.v023.i07`.

Stasinopoulos M, Rigby RA, Heller GZ, Voudouris V, De Bastiani F (2017). "The GAMLSS family of distributions." In *Flexible regression and smoothing using GAMLSS in R*, pp. 153–189. CRC Press. ISBN 9781138197909.

Affiliation:

Jaime Mosquera Gutiérrez
Universidad Nacional de Colombia
School of Statistics
Faculty of Sciences
Universidad Nacional de Colombia
Cra. 65 #59a-110
Medellín, Colombia
E-mail: jmosquerag@unal.edu.co