AdequacyModel: An R Package for Probability Distributions and General Purpose Optimization

Pedro R. D. Marinho [1*], Rodrigo B. Silva[1†], Marcelo Bourguignon[2§], Gauss M. Cordeiro [3‡], Saralees Nadarajah[4✱],

1 Department of Statistics, Federal University of Paraíba, João Pessoa, Paraíba, Brazil
2 Department of Statistics, Federal University of Rio Grande do Norte, Natal, Rio Grande do Norte, Brazil
3 Department of Statistics, Federal University of Pernambuco, Recife, Pernambuco, Brazil
4 School of Mathematics, University of Manchester, Manchester, United Kingdom

* pedro.rafael.marinho@gmail.com

Abstract

The paper deals with the `AdequacyModel` computational library version 2.0.0 implemented in the `R` language. The library has been used and referenced in various papers in the area of probability and statistics. This library has been shown to be very useful for researchers in the area of probability / statistics and serves as the basis for the `Newdistns` library, version 2.1 published in an impact journal in the area of computational statistics, see https://cran.r-project.org/web/packages/Newdistns/. The library `AdequacyModel` also serves as the basis of the library `Wrapped` (version 2.0) that is deposited in https://cran.r-project.org/web/packages/Wrapped/.
In addition, the `AdequacyModel` library, version 2.0.0 presents an interesting implementation of the Particle Swarm Optimization - PSO method with a minor modification in the original algorithm. This method has proved very useful for maximizing log-likelihood functions with complex search regions. In new classes of probabilistic models that have been proposed in the last years, methods of Newton and quasi-Newton have not been shown efficient to optimize these functions.
The introduction of new lifetime models has been played an important role in the treatment of survival data. For some of these new models, however, it is quite difficult to obtain the maximum likelihood estimators due to, among other factors, evidence of flat regions in the search space. It makes several well-known derivative-based optimization tools unsuitable for obtaining such estimates. To circumvent this problem, we introduce the `AdequacyModel` package for the `R` statistical computing environment with two major contributions, namely: a general purpose optimization method based on the Particle Swarm Optimization approach and a set of statistical measures for the assessment of the adequacy of lifetime models for a given dataset. We provide a greater control of the optimization process by introducing a stop criterion which is based on a minimum number of iterations and the variance of a given proportion of optimal values. It is important to emphasize that the proposed `R` package can be used not only in statistics but in physics and mathematics as demonstrated in several examples throughout the paper. Details and contributions to the development of the package may be presented and monitored on GitHub; see https://github.com/prdm0/AdequacyModel.

1 Introduction

In survival analysis, practitioners are usually interested in choosing the model that provides the best fit from a broad class of candidate models. In this sense, lifetime distributions are continually evolving in parallel with computer-based tools, which allow for the use of more complex distributions with a larger number of parameters to better study sizable masses of data. The last two decades have been very prolific in proposing new parametric models for lifetime data analysis and a great number of methods that generate new distributions can be found in the literature. In addition to extending the traditional models, the relevance of these new distributions relies on the fact that, according to the problem, each of them can better fit a given data set. For a survey on the most important recent lifetime distributions the reader are referred to [1] and [2].
The main concern about recent proposed models is that in several cases one can obtain different solutions from different initial values when optimizing the respective likelihood functions, indicating the presence of flat regions in the search space. The term "flat" is used here to indicate that the minimum modulus of a function in a region is (in some sense) of the same order as the maximum modulus. In this case, most derivative-based optimization tools usually encounter difficulties such as getting trapped in local minima, which makes such approaches unsuitable to obtain the corresponding maximum likelihood estimates (MLEs). This is not, however, an exclusive problem of recent lifetime models. Several univariate and multivariate functions present the same issue. To circumvent this problem, several optimization algorithms based on swarm intelligence have been proposed over the last decades. This class of methods have shown efficiency and robustness, although simple to implement. One of very popular swarm intelligence methods is the Particle Swarm Optimization (PSO) for finding optimized solutions. PSO is a stochastic search method introduced by [3]. It is based on simple social behavior exhibited by birds and insects and, due to its simplicity in implementation, it has gained great popularity in the optimization community. It has great advantages: high level of convergence and computational cost, if compared with other heuristic search methods. It traditionally uses a random sampling to find the optimums, but it is superior, if compared with derivative-based methods, when the information about localization of the minimum (or maximum) is poor, which is the case when we have functions with flat regions. Further details on the philosophy of PSO method can be found in [4].
Several variants of PSO algorithm have been proposed in the literature in order to fit different types of problems. [5] proposed a mirror-extended Curvelet transform and PSO to solve the problem of speckle noise and low contrast in Synthetic Aperture Radar images. Since data mining demands fast and precise partitioning of large datasets, it usually comes with a wide range of attributes or features, which requires serious computational restrictions on the relevant clustering techniques. [6] presents an overview of PSO techniques for cluster analysis. The issue of choosing the most adequate values in the Support Vector Machine (SVM) methodology can be structured in terms of an optimization problem in order to minimize a prediction error. [7] proposed an integrated PSO algorithm (PSO+SVM) to solve this problem. [8] presents a PSO overview under a Bayesian perspective, providing a formal framework for incorporation of prior knowledge about the problem that is being solved. [9] present the maximum likelihood estimation via PSO algorithm to estimate the mixture of two Weibull parameters with complete and multiple censored data.
The chief goal of our paper is twofold. First, we introduce a variation of PSO method in the `AdequacyModel` package for the `R` statistical computing environment [10], which provides a robust optimization method for obtaining maximum likelihood estimates (MLEs) for lifetime distributions, in special those with approximately flat regions.

The AdequacyModel package provides a greater control of the optimization process by introducing a stop criterion that is based on a minimum number of iterations and the variance of a given proportion of optimal values. The numerical evidence shows that the proposed optimization method typically gives more reliable results then those based on derivatives, such as the quasi-Newton Broyden-Fletcher-Goldfarb-Shanno (BFGS), Nelder-Mead and simulated-annealing (SANN) methods. For comparison purposes, we use several functions that are well-known by their optimization difficulties. Second, we provide several measures to evaluate the adjustment quality of competing statistical models for a given data set. Even though our focus lies in lifetime models, the optimization package we propose can be used in several other areas as demonstrated in some examples throughout the paper.
The remainder of this paper is as follows. Section 2 introduces some theoretical background of swarm intelligence and PSO paradigms, which contains the general ideas underlying the PSO approach. Section 3 presents PSO algorithm designed for AdequacyModel R package. Section 4 develops the practical examples through which the effectiveness of the proposed PSO algorithm is challenged to the results obtained from other techniques, especially those based on derivatives. Section 5 contains an application that uses real (not simulated) data. Finally, Section 6 gives some concluding remarks on the main findings of the paper and the current package usage.

2 Conceptual design of the framework

2.1 Swarm intelligence

Swarm intelligence is an exciting research field still in its infancy if compared to other paradigms in artificial intelligence. It is a branch of artificial intelligence concerned to the study of collective behavior of decentralized and self-organized systems in a social structure. These kind of systems are composed by agents that interact in a small organization (swarm) wherein each individual is a particle. The main idea behind swarm intelligence is that an isolated particle has a very limited action in search an ideal point for the solution of an nondeterministic polynomial time (NP)-complete problem. However, the joint behavior of the particles in the search region shows evidence of artificial intelligence, i.e., the ability to take decisions to respond to changes. In this sense, the swarm intelligence concept arises directly from nature and is based on, for example, the self-organizing exploratory pattern of the schools of fish, flocks of birds and ant colonies. This collective behavior can not be described simply by aggregating the behavior of each element. Such situations have encouraged practitioners to obtain a satisfactory effect in the search for solutions to complex problems by studying methods that promote intelligent behavior through collaboration and competition among individuals. Swarm-based algorithms have been widely developed in the last decade and the many successful applications in a variety of complex problems make it a very promising, efficient and robust optimization tool, although very simple to implement. The idea is modeling very simple local interactions among individuals from which complex problem-solving behaviors arise.

2.2 Proposed particle swarm optimization

The PSO algorithm is conceptually based on the social behavior of biological organisms that move in groups, such as birds and fishes. It has been provided good solutions for problems of global function optimization with box-constrained. The fundamental component of the PSO algorithm is a particle, which can move around in the search space in direction of an optimum by making use of its own information as well

as that obtained from other particles within its neighborhood. The performance of a particle is affected by its fitness, which is evaluated by calculating the objective function of the problem to be solved. The particles movement in the search space is randomized. For each iteration of the PSO algorithm, the leader particle is set by minimizing the objective function in the corresponding iteration. The remaining particles arranged in the search region will randomly follow the leader particle and sweep the area around the leader particle. In this local search process, another particle may become the new leader and the other particles will follow the new leader randomly.
Mathematically, a particle i is featured by three vectors, namely:

- Its current location in the n-dimensional search space denoted by $\boldsymbol{x}_i = (x_{i1}, \dots, x_{in})$.
- The best individual position it has held so far denoted by $\boldsymbol{p}_i = (p_{i1}, \dots, p_{in})$.
- Its velocity $\boldsymbol{v}_i = (v_{i1}, \dots, v_{in})$.

Usually, the current locations x_i and velocities $\boldsymbol{v}_i$ are initialized by sampling from uniform distributions throughout the search space and by setting a maximum velocity value $v_{\max}$, respectively. Then, the particles move over the search space in sequential iterations driven by the following set of update equations:

- $v_{i,d}(t+1) = v_{i,d}(t) + c_1\, r_1\, [p_{i,d}(t) - x_{i,d}(t)] + c_2\, r_2\, [p_{g,d}(t) - x_{i,d}(t)]$.
- $x_{i,d}(t+1) = x_{i,d}(t) + v_{i,d}(t+1), \quad d = 1, \dots, n,$

where c_1 and c_2 are constants, r_1 and r_2 are independent uniform random numbers generated at every update along each individual direction $d = 1, \dots, n$ and $p_g(t)$ is the n-dimensional vector of the best position encountered by any neighbor of particle i. Note that velocities and positions at time $t+1$ are influenced by the distances of the particle's current location from its individual best historical experience $p_i(t)$ and its neighborhoods best historical experience $p_g(t)$, in a cooperative way.
The proposed PSO algorithm is a small modification of the standard algorithm of the PSO method proposed by [3], where $f : \mathcal{R} \mapsto \mathbb{R}$, with $\mathcal{R} \subseteq \mathbb{R}^n$, is the objective function to be minimized, S is the number of particles of the swarm (set of feasible points), each particle having a location vector $x_i \in \mathcal{R}$ in the search space and a velocity vector defined by $v_i \in \mathcal{R}$. Let p_i be the best known position of particle i and g the best position of all particles. The small modifications were highlighted in the standard algorithm below. The default optimization did not address the optimization problem restricted to a region $\mathcal{R}$. In the course of the iterations, it is common for several particles to fall outside the search region $\mathcal{R}$. The strategy of eliminating these particles and randomly relocating them in the search region increases the variability of the algorithm by preventing all particles from converging to a local minimum.
1. For each particle $i = 1, \dots, S$ do:

- Initialize the particle's position with a uniformly distributed random vector: $x_i \sim U(b_{lo}, b_{up})$, where b_{lo} and b_{up} are the lower and upper boundaries of the search-space.
- Initialize the particle's best known position to its initial position: $p_i \longleftarrow x_i$.
- If $f(p_i) < f(g)$ update the swarm's best known position: $g \longleftarrow p_i$.
- Initialize the particle's velocity: $v_i \sim U(-|b_{up} - b_{lo}|, |b_{up} - b_{lo}|)$.

2. Until a termination criterion is met (e.g. number of iterations performed, or a solution with adequate objective function value is found), repeat:

- For each particle $i = 1, \dots, S$ do:

```
    – Pick random numbers: $r_p, r_g \sim U(0,1)$.
    – For each dimension $d = 1, \ldots, n$ do:
        ∗ Update the particle's velocity: $v_{i,d} \longleftarrow \omega\, v_{i,d} + \varphi_p r_p (p_{i,d} - x_{i,d}) + \varphi_g r_g (g_d - x_{i,d})$.
    – Update the particle's position: $x_i \longleftarrow x_i + v_i$
    – If $x_i \notin \mathcal{R}$
        ∗ Eliminate $x_i$.
        ∗ Generate new values $x_i \in \mathcal{R}$ (random values).
    – If $f(x_i) < f(p_i)$ do:
        ∗ Update the particle's best known position: $p_i \longleftarrow x_i$
        ∗ If $f(p_i) < f(g)$ update the swarm's best known position: $g \longleftarrow p_i$.
3. Now $g$ holds the best found solution.
```

The parameter ω is called inertia coefficient and, as the name implies, controls the inertia of each particle arranged in the search space. The quantities φ_p and φ_g control the acceleration of each particle and are called acceleration coefficients. The PSO algorithm described above implemented in the programming language `R` is presented in the next section. A conditional stopping criterion will be discussed.

The choice of constants ω, φ_p and φ_g can dramatically affect the performance of the algorithm in the optimization process. Discussions about appropriate parameter choices have been the subject of research. Some references include: [4] and [11].

3 The `AdequacyModel` `R` package

3.1 Multi-parameter global optimization

The above algorithm is implemented in the `AdequacyModel` package available on the website of `R`. It is quite general and can be applied to maximize or minimizing any objective function involving or not a database, taking into account restriction vectors delimiting the search space. We want to make clear that the pso function is constructed to minimize an objective function. However, note that to maximize f is equivalent to minimize $-f$. A brief description of the `AdequacyModel` package arguments is listed below:

- `func`: an objective function to be minimized;
- `S`: number of particles considered. By default, the number of particles is equal to 150;
- `lim_inf` e `lim_sup`: define the inferior and superior boundaries of the search space, respectively;
- `e`: error considered. The algorithm stops if the variance in the last iterations is less than or equal to `e`;
- `data`: by default `data = NULL`. However, when the `func` is a log-likelihood function, `data` is a data vector;
- `N`: minimum number of iterations (default `N = 500`);

- `prop`: Proportion of last optimal values whose variance is calculated, which is used as a stop criterion. That is, if the number of iterations is greater than or equal to the minimum number of iterations `N`, then calculate the variance of the obtained last optimal values, where $0 \leq$ `prop` ≤ 1.

One advantage of the PSO method is that we do not need to concern with initial values. Problems with initial values are frequent in methods such as the BFGS when the objective function involves flat or nearly flat regions. Depending on the chosen initial values we can obtain totally different results. This kind of issue is not usual in heuristic-based methods, in which the updated steps include randomness (generation of pseudo-random numbers). The following example presents issues related to the initial guesses for the algorithm and shows how to use the `pso` function, especially the usage of argument `func` to specify the objective function. In order to provide a greater control of the optimization process, we define N as the stop criterion that states the minimum number of iterations. The number of optimal values considered in the variance calculation is given by the proportion of optimal values stated by argument prop, which is equal to 0.2 by default. In other words, if the 20% last optimal values show variance less than or equal to e, the algorithm will stop the global search, indicating convergence according to the stated criteria. These stop criteria indicate that there is no significant improvements in the global search for this proportion of iterations. Thus, if the variance is less than or equal to $\varepsilon > 0$ assigned to the argument e of the `pso` function, the algorithm will stop the iterations and return the best point that minimizes the objective function.

3.2 Examples

3.2.1 Trigonometric function

Initially, we will consider the case of a global search in a univariate function, where we are interested in estimate a one-dimensional vector. Consider the objective function $f(\theta) = 6 + \theta^2 \sin(14\theta)$. This is a function with some local minima, such that the value of θ that globally minimizes f is equal to 2.3605 and $f(2.3605) = -11.5618$. Figure 1 shows the curve of $f(\theta)$, for $\theta \in [-2.5, 2.5]$. The blue square symbol indicates the global minimum obtained by BFGS, SANN and Nelder-Mead methods. The red bullet in turn represents the global minimum obtained by the PSO method.

```
R> f <- function(x){
+      -(6 + x^2 * sin(14*x))
+ }
R> f_pso <- function(x,par){
+ theta = par[1]
+      -(6 + theta^2 * sin(14*theta))
+ }
R> set.seed(9)
R> result_pso_f = pso(func = f_pso, S = 500, lim_inf = c(-2.5),
+                      lim_sup = c(2.5), e = 0.0001)
R> set.seed(9)
R> result_sann_f = optim(par=c(0), fn = f, lower = -2.5, upper = 2.5,
+                         method = "SANN")
R> result_bfgs_f = optim(par=c(0), fn = f, lower = -2.5, upper = 2.5,
+                         method = "BFGS")
R> result_nelder_f = optim(par=c(0),fn = f, lower = -2.5, upper = 2.5,
+                           method = "Nelder-Mead")
```

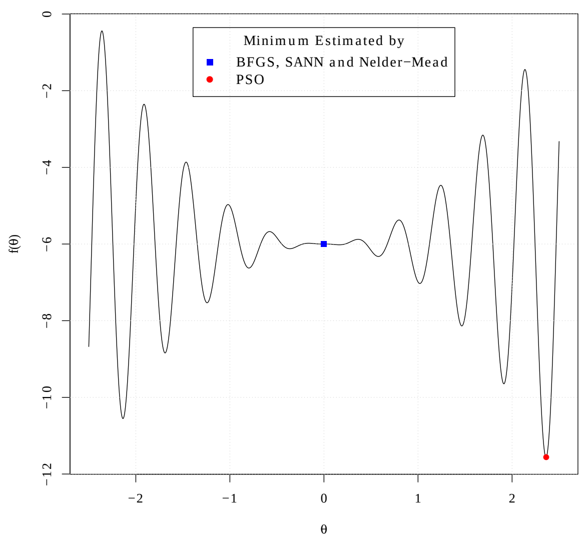

Fig 1. Function $f(\theta) = 6 + \theta^2 \sin(14\theta)$ with global minimum estimates.

Note that the global minimum estimate obtained by the BFGS, SANN and Nelder-Mead methods through the `optim` function (for more details, execute `?optim`) are heavily influenced by initial value 0. From Figure 1, it is quite clear that there is $\varepsilon > 0$ such that f has derivative close to 0 around $(-\varepsilon, \varepsilon)$. On the other hand, the `pso` function from `AdequacyModel` package provides the true global minimum, which obviously coincides with the analytic solution. Notice that all evaluated methods converge according to their respective stop criteria. For the BFGS, SANN and Nelder-Mead methods, we set the same initial value 0. In the case of SANN and `pso` functions, which involve randomization, we set a seed equal to 9, i.e. `set.seed(9)`. The global minimum values obtained by the BFGS, Nelder-Mead and SANN methods are identical and influenced by the starting values. Unlike these methodologies, the PSO method implemented by the `pso` function does not require initial values. These results can be replicated using the `AdequacyModel` package and the code below:

```
R> f <- function(x){
+     -(6 + x^2 * sin(14*x))
+ }
R> f_pso <- function(x,par){
+ theta = par[1]
+     -(6 + theta^2 * sin(14*theta))
+ }
R> set.seed(9)
R> result_pso_f = pso(func = f_pso, S = 500, lim_inf = c(-2.5),
+                     lim_sup = c(2.5), e = 0.0001)
R> set.seed(9)
R> result_sann_f = optim(par = c(0), fn = f, lower = -2.5, upper = 2.5,
+                        method = "SANN")
R> result_bfgs_f = optim(par = c(0), fn = f, lower = -2.5, upper = 2.5,
+                        method = "BFGS")
R> result_nelder_f = optim(par = c(0), fn = f, lower = -2.5, upper = 2.5,
+                          method = "Nelder-Mead")
```

The use of the `pso` function is rather simplistic. This function is implemented to be parsimonious in order to facilitate its use. The following example uses `pso` function for multi-parameter optimizations.

3.2.2 Easom function

Consider the Easom function $f(x,y) = -\cos(x)\cos(y)\exp\{-[(x-\pi)^2 + (y-\pi)^2]\}$, for $-10 \leq x,\, y \leq 10$. Some plots are displayed at different angles in Figures 2(a) and 2(b). The Easom function is minimized at $x = y = \pi$, and $f(\pi,\pi) = -1$. The use of the `pso` function to minimize the above function is

```
R> easom <- function(x,par){
+     x1 = par[1]
+     x2 = par[2]
+     -cos(x1) * cos(x2) * exp(-((x1-pi)^2 + (x2-pi)^2))
+ }
R> set.seed(9)
R> results_pso = pso(func = easom, S = 500, lim_inf = c(-10,-10),
+                      lim_sup = c(10,10), e = 0.0001)
```

Before execution of `pso` function, we set `set.seed(9)`, for which the same results can be replicated. The estimated minimum points by the `pso` function are $\hat{x} = 3.139752$ and $\hat{y} = 3.141564$, which are very close to $x = y = \pi$. The convergence of the methodology for very close values to the global optimum can be best observed in Easom function levels curves displayed in Figure 3.

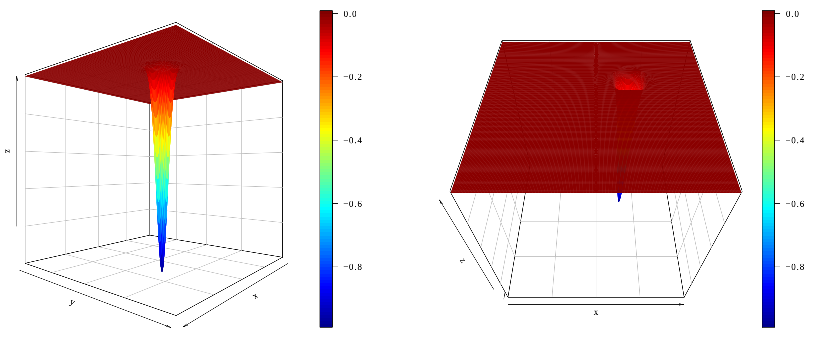

Fig 2. Easom function at two different angles.

We use the BFGS method through the `optim` function and set the initial values as $x = -9$ and $y = 9$. Note that the convergence is achieved in the BFGS method and the estimated minimum points coincides with the fixed initial values ($\hat{x} = -9$ and $\hat{y} = 9$), which is quite different from the minimum true point $x = y = \pi$, showing that this method is very sensitive to initial values. The reader can observe this fact from the code below.

```
R> easom1 <- function(x){
+ x1 = x[1]
+ x2 = x[2]
+ -cos(x1) * cos(x2) * exp(-((x1-pi)^2 + (x2-pi)^2))
+ }
R> result_bfgs_easom = optim(par = c(9,9), fn = easom1, method = "BFGS")
```

Notice that `result_bfgs_easom$convergence == 0` is equal to `TRUE`, which indicates convergence. Execute `help(optim)` for more details about the convergence criterion of the BFGS method implemented in the `optim` function. In the case of Easom function, the convergence is harmed by the existence of infinite candidates to the minimum point distributed on a flat region. The output stored in the object `result_bfgs_easom` is presented below:

```
R> result_bfgs_easom
$par
[1] -9  9

$value
[1] -1.283436e-30

$counts
function gradient
1        1

$convergence
[1] 0

$message
NULL
```

Setting `result_nelder_easom = optim(par = c(-9,9), fn = easom1, method = "Nelder-Mead")`, we also obtain a distant estimated point from the true global minimum point, where $\hat{x} = -8.1$ and $\hat{y} = 9$ give a minimum value approximately equal to zero. The results stored in `result_nelder_easom` are given below:

```
R> result_nelder_easom
$par
[1] -8.1  9.0

$value
[1] -3.609875e-71

$counts
function gradient
3       NA

$convergence
[1] 0

$message
NULL
```

[3] A similar fact is obtained using the simulated method in which the estimates can be obtained with the code that follows:

```
R> set.seed(9)
R> result_sann_easom = optim(par = c(-9,9), fn = easom1, method = "SANN")
```

As in the previous cases, it is noted that `result_sann_easom$convergence == 0` is `TRUE` (there is convergence) and the estimated minimum point has coordinates distant from the coordinates of the true minimum point, where the estimated coordinates are $\hat{x} = 1.110688$ and $\hat{y} = 13.934928$ with seed fixed at 9, i.e. `set.seed(9)`.

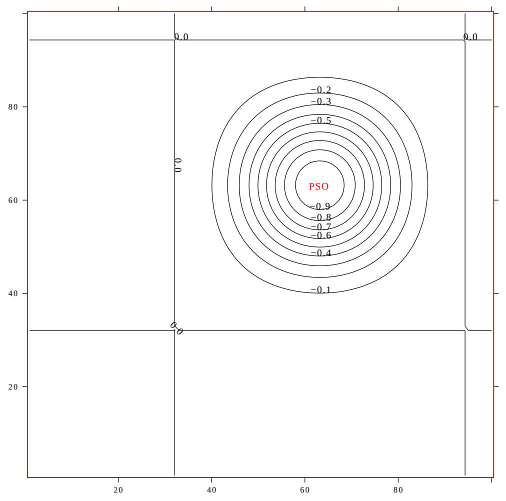

Fig 3. Curves of levels of the Easom function.

3.2.3 Cross-in-tray function

Now, we consider the example of the use of `pso` function to minimize the Cross-in-tray function. This is a difficult function to be minimized for different reasons from those presented in the previous example. The Cross-in-tray function has many local minima as can be seen in Figures 4(a) and 4(b). This fact can certainly harm the convergence of various algorithms that search for a global optimum. The Cross-in-tray function is defined by

$$f(x, y) = -0.0001\left(\left|\sin(x)\sin(y)\exp\left(\left|100 - \frac{\sqrt{x^2+y^2}}{\pi}\right|\right)\right| + 1\right)^{0.1},$$

where $-10 \leq x,\, y \leq 10$ and

$$\text{Min} = \left\{\begin{array}{ccc} f(1.34941,\,-1.34941) & = & -2.06261 \\ f(1.34941,\,1.34941) & = & -2.06261 \\ f(-1.34941,\,1.34941) & = & -2.06261 \\ f(-1.34941,\,-1.34941) & = & -2.06261 \end{array}\right.$$

Note that this function has four points of global minimum. Any estimate of the minimum points $(\hat{x},\, \hat{y})$ that applied in $f(\cdot)$ presents minimum value close to -2.0626 will be a good solution.

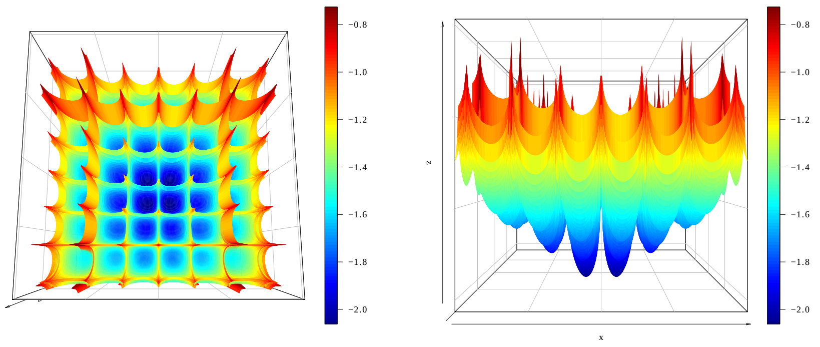

Fig 4. Cross-in-tray function at two different angles.

By means of the `optim` function, we note the convergence of the BFGS, SANN and Nelder-Mead methods with initial values at $x = 0$ and $y = 0$ and estimated values of x and y equal to $\hat{x} = \hat{y} = 0$ in the three approaches, with $f(\hat{x}, \hat{y}) = -0.0001$. The minimization of the Cross-in-tray function by using the PSO function achieves a satisfactory outcome as we can see in Figure 5. By the `pso` function, the estimated minimum point is $(1.3490, 1.3490)$ with a minimum value equal to $f(1.3490, 1.3490) = -2.0626$. These same results can be obtained with the script below:

```
R> cross <- function(x,par){
+   x1 = par[1]
+   x2 = par[2]
+   -0.0001 * (abs(sin(x1) * sin(x2) * exp(abs(100-sqrt(x1^2+x2^2)/pi)))
+ + 1)^0.1
+ }
R> set.seed(9)
R> result_pso_cross <- pso(func = cross, S = 500, lim_inf = c(-10,-10),
+                          lim_sup = c(10,10), e = 0.0001)
```

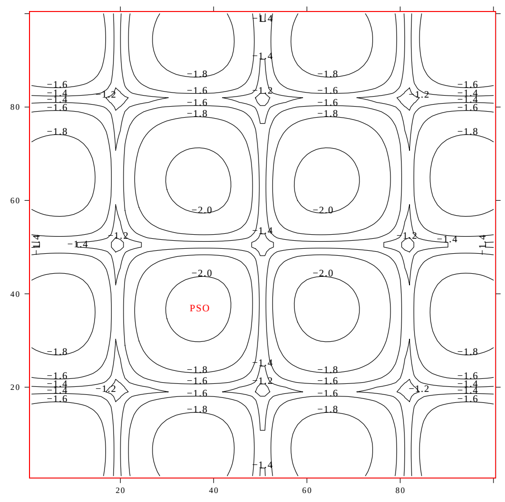

Fig 5. Curves of levels of Cross-in-tray function.

Note: The results of the optimization using the `optim` function and the Nelder-Mead, BFGS and simulated annealing methods can be obtained with the code below such that, for all these methodologies, the initial shot is given at the point $(0, 0)$.

```
R> cross1 <- function(x){
+     x1 = x[1]
+     x2 = x[2]
+     -0.0001 * (abs(sin(x1) * sin(x2) *
+     exp(abs(100-sqrt(x1^2+x2^2)/pi))) + 1)^0.1
+ }
R> result_bfgs_cross = optim(par = c(0,0), fn = cross1, lower = -10,
+                            upper = 10, method = "BFGS")
R> result_nelder_cross = optim(par = c(0,0), fn = cross1, lower = -10,
+                              upper = 10, method = "Nelder-Mead")
R> set.seed(9)
R> result_sann_cross = optim(par = c(0,0), fn = cross1,
+                            lower = -10, upper = 10,
+                            method = "SANN")
```

3.2.4 Hölder function

Consider the case of the Hölder function, very peculiar and difficult to be optimized. It is defined by

$$f(x, y) = -\left|\sin(x)\cos(y)\exp\left(\left|1 - \frac{\sqrt{x^2+y^2}}{\pi}\right|\right)\right|,$$

where

$$\text{Min} = \begin{cases} f(8.05502,\, 9.66459) & = & -19.2085 \\ f(-8.05502,\, 9.66459) & = & -19.2085 \\ f(8.05502,\, -9.66459) & = & -19.2085 \\ f(-8.05502,\, -9.66459) & = & -19.2085, \end{cases}$$

where $-10 \le x,\, y \le 10$. Figure 6 displays the plots of the Hölder function defined above.

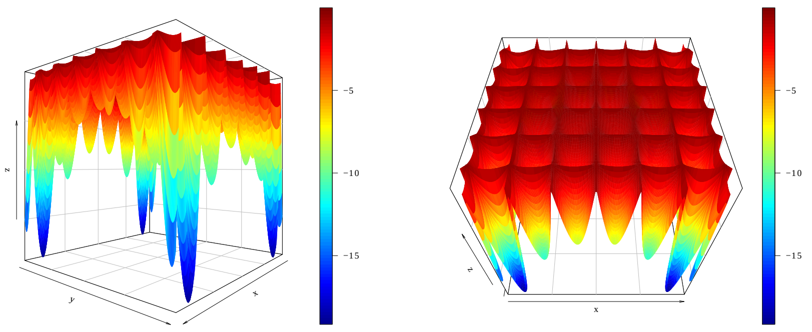

Fig 6. Hölder function at two different angles.

For the Hölder function, the results obtained from BFGS, SANN and Nelder-Mead methods, as in the previous examples, were not good. However, in all cases, there was a convergence following these methodologies implemented in the `optim` function. With initial values at the point $(0, 0)$, the convergence leads to this point, i.e., the three methodologies estimate the minimum point at $\hat{x} = 0$ and $\hat{y} = 0$. For the SANN method, we set `set.seed(9)`. An interesting fact is that the `pso` function also failed to get good estimates for `S = 500`, i.e., when considering 500 particles for optimization. However, the problem is easily circumvented by raising the number of particles. Figure 7 displays plots of the levels of the Hölder function with the point of convergence of the PSO algorithm. This result was obtained by using the following script:

```
R> holder <- function(x,par){
+ x1 = par[1]
+ x2 = par[2]
+ -abs(sin(x1)*cos(x2) * exp(abs(1 - sqrt(x1^2+x2^2)/pi)))
+ }
R> result_pso_holder = pso(func = holder, S = 10000, lim_inf = c(-10,-10),
+                           lim_sup = c(10,10), e = 0.0001)
```

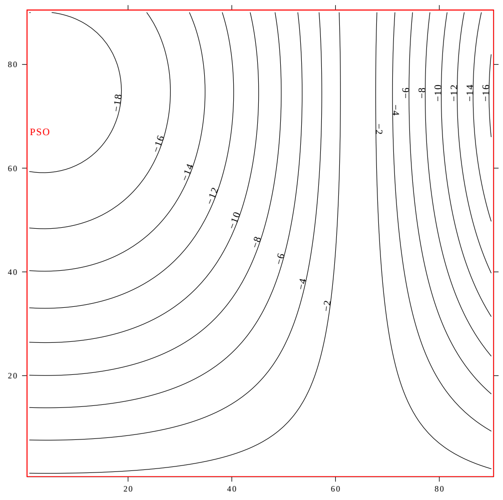

Fig 7. Curves of levels of Hölder function.

4 Fitting distributions with `AdequacyModel`

In inference, the problem of deciding on the suitability of an unknown distribution F_θ, from a sample $X_1, \ldots, X_n$ is equivalent to the decision problem on an unknown parameter θ. Let $\mathcal{F} = \{F_\theta;\, \theta \in \Theta\}$ a family of distributions, where Θ is the parameter space of θ. By assumption, it is admitted that in $\mathcal{F}$ exists a F_θ such that F evaluated in $\hat{\theta}_n$. The best element F_θ in $\mathcal{F}$ is obtained by obtaining the maximum likelihood estimator (MLE) $\hat{\theta}_n$ of θ.
However, not always the assumption that $\mathcal{F}$ is adequate. Thus, we need statistics to verify the adequacy of the distribution to the observations, that is, we need test the adherence for the data. Alternatives to the likelihood ratio test were proposed by [12], which are corrections to the Carmér-von Mises (W^*) and Anderson Darling (A^*) statistics. We use these statistics when we have a random sample $x_n = \{x_1, \ldots, x_n\}$ with empirical distribution function $F_n(x)$ and we want to test if the sample has a

specified distribution. The W^* and A^* are, respectively, given by

$$W^* = \left\{ n \int_{-\infty}^{+\infty} \{F_n(x) - F(x;\widehat{\theta}_n)\}^2 dF(x;\widehat{\theta}_n) \right\} \left(1 + \frac{0.5}{n}\right) = W^2 \left(1 + \frac{0.5}{n}\right), \quad (1)$$

$$\begin{aligned} A^* &= \left\{ n \int_{-\infty}^{+\infty} \frac{\{F_n(x) - F(x;\widehat{\theta}_n)\}^2}{\{F(x;\widehat{\theta})(1 - F(x;\widehat{\theta}_n))\}} dF(x;\widehat{\theta}_n) \right\} \left(1 + \frac{0.75}{n} + \frac{2.25}{n^2}\right) \\ &= A^2 \left(1 + \frac{0.75}{n} + \frac{2.25}{n^2}\right), \end{aligned} \quad (2)$$

where $F_n(x)$ is the empirical distribution function, $F(x;\widehat{\theta}_n)$ is the postulated cdf evaluated at the MLE $\widehat{\theta}_n$ of θ, and W^2 and A^2 are the Cramér-von Mises and Anderson-Darling statistics, respectively. This statistic is given by the difference between $F_n(x)$ and $F(x;\widehat{\theta}_n)$. Lower values of statistics provide further evidence that $F(x;\widehat{\theta}_n)$ generate the data. The null hypothesis tested using the statistics (1) and (2) is that the random sample has distribution $F(x;\theta)$. According to [12], the W^2 and A^2 statistics can be readily calculated as

$$W^2 = \sum_{i=1}^{n} [u_i - \{(2i-1)/(2n)\}]^2 + 1/(12n) \quad (3)$$

and

$$A^2 = -n - n^{-1} \sum_{i=1}^{n} \{(2i-1)\log(u_i) + (2n+1-2i)\log(1-u_i)\}, \quad (4)$$

where $u_i = \Phi((y_i - \overline{y})/s_y)$, $v_i = F(x_i;\widehat{\theta}_n)$, $y_i = \Phi^{-1}(v_i)$ (Φ is the standard normal cdf) and s_y is the sample standard deviation of the y_i's, for $i = 1, 2, \ldots, n$. The algorithm below can be adopted to obtain W^* and A^*:

1. Estimate θ by $\widehat{\theta}_n$, order the sample values in crescent values to calculate $v_i = F(x_i;\widehat{\theta}_n)$;

2. Calculate $y_i = \Phi^{-1}(v_i)$, where Φ^{-1} is the standard normal quantile function;

3. Calculate $u_i = \Phi\{(y_i - \overline{y})/s_y\}$, where $\overline{y} = n^{-1}\sum_{i=1}^{n} y_i$ and $s_y^2 = (n-1)^{-1}\sum_{i=1}^{n}(y_i - \overline{y})^2$;

4. Calculate W^2 e A^2 using equations (3) and (4), respectively;

5. Obtain $W^* = W^2(1 + 0.5/n)$ and $A^* = A^2(1 + 0.75/n + 2.25/n^2)$, where n is the sample size;

6. We reject $\mathcal{H}_0$ at the significance level α if the test statistics exceed the critical values presented by [12].

What is commonly done in practice is to use W^* and A^* to compare two or more continuous distributions. The distribution that gives the lowest values of W^* or A^* is the best suited to explain the random sample. The `goodness.fit` function provides some useful statistics to assess the quality of fit of probabilistic models, including the W^* and A^* statistics. The function can also determine other goodness-of-fit statistics such as the AIC (Akaike Information Criterion), CAIC (Consistent Akaikes Information Criterion), BIC (Bayesian Information Criterion), HQIC (Hannan-Quinn Information Criterion) and KST (Kolmogorov-Smirnov Test). The general form for the function is given below with the descriptions of each one of its arguments.

```
goodness.fit(pdf, cdf, starts = NULL, data, method = "PSO",
             domain = c(0,Inf), mle = NULL)
```

where

- `pdf`: probability density function;
- `cdf`: cumulative distribution function;
- `starts`: initial parameters to maximize the likelihood function;
- `data`: data vector;
- `method`: method used for minimization of the -log-likelihood function. The methods supported are: PSO (default), BFGS, Nelder-Mead, SANN, CG (conjugate gradients). We can also provide only the first letter of the methodology, i.e., P, B, N, S or C, respectively;
- `domain`: domain of the pdf. By default the domain of the pdf is the open interval $(0, \infty)$. This option must be a vector with two components;
- `mle`: vector with the MLEs. This option should be used if one already has knowledge of the MLEs. The default is NULL, i.e., user the function will try to obtain the MLEs;
- `...`: If `method = "PSO"`, then all the arguments of `pso` function could be passed to the `goodness.fit` function.

An important observation is that it is not necessary to define the likelihood function or log-likelihood. Just we need to define the pdf and cdf. The function will self-criticism to the arguments passed to the `goodness.fit`. For example, if supplied to the arguments `pdf` or `cdf` functions that do not be pdfs and cdfs, a notice will be given so that the user can check the arguments passed. We provide below two examples of the use of the `goodness.fit` function.

4.1 Carbon fiber data

Suppose the problem is that it has a data set of stress (until fracture) of carbon fibres (in Gba). The data were obtained by [13] and are available for use in the `AdequacyModel` package and can be accessed with the command `data(carbone)`. Further, details regarding the set of data is obtained in the documentation of the package with the command `help(carbone)`. Suppose also that we are interested in obtaining the best model in $\mathcal{F} = \{F_\theta;\, \theta \in \Theta\}$ that can represent the distribution of $X_1, \ldots, X_n$, whose observations are in `carbone`. Here, we consider that $\mathcal{F}$ is the Exp-Weibull (exponentiated Weibull) distribution. Its cdf is given by

$$F(x, \alpha, \beta, a) = \left\{1 - \exp\left[-(\alpha x)^{\beta}\right]\right\}^{a},$$

where α, β and c are positive parameters and $x > 0$. Thus, each element in $\mathcal{F}$ is of the form $F(x; \alpha, \beta, a)$. We initially implement the pdf $f(x; \alpha, \beta, a)$ and the cdf $F(x; \alpha, \beta, a)$. They will serve as arguments for `pdf` and `cdf`, respectively. We present below the implementation of the functions that will be given to the `goodness.fit` function.

```
R> # Probability density function.
R> pdf_expweibull <- function(par, x) {
+     alpha = par[1]
+     beta = par[2]
+     a = par[3]
+     alpha * beta * a * exp(-(alpha * x) ^ beta) * (alpha * x) ^ (beta
+     - 1) * (1 - exp(-(alpha * x) ^ beta)) ^ (a - 1)
+ }
R> # Cumulative distribution function.
R> cdf_expweibull <- function(par, x) {
+     alpha = par[1]
+     beta = par[2]
+     a = par[3]
+     (1 - exp(-(alpha * x) ^ beta)) ^ a
+ }

R> data(carbone)
R> results = goodness.fit(pdf = pdf_expweibull, cdf = cdf_expweibull,
+  starts = c(1, 1, 1), data = carbone, method = "BFGS",
+  domain = c(0, Inf), mle = NULL)
```

The object `results` feature all the goodness-of-fit statistics cited previously as well as the MLEs in case of `mle = NULL` (default). The error of the MLEs if the argument `method` receives `PSO`, `BFGS`, `Nelder-Mead`, `SANN` and `CG`. Thus,

- `R> results$W` provides the statistic W^*;
- `R> results$A` provides the statistic A^*;
- `R> results$KS` provides the statistic of Kolmogorov-Smirnov;
- `R> results$mle` provides a vector with the MLEs of the model parameters given as arguments for the `pdf`;
- `R> results$AIC`: provides the AIC statistic;
- `R> results$CAIC`: provides the CAIC statistic;
- `R> results$BIC`: provides the BIC statistic;
- `R> results$HQIC`: provides the HQIC statistic;
- `R> result$KS`: returns an object of class `htest` with information on the Kolmogorov-Smirnov test;
- `R> results$Erro`: provides the standard errors of the MLEs of the parameters, which index the model parameters given as arguments for the `pdf` and `cdf`;
- `R> results$value`: displays the minimum value of the function `-log(likelihood)`;
- `R> result$Convergence`: provides information on the convergence of the method passed as an argument for `method`. If `result$Convergence` == 0 for `TRUE`, there was convergence.

In case of `method = "PSO"` (default), the standard errors will not be provided. The researcher may obtain such errors through bootstrap. For more details of how to obtain the MLE standard errors via bootstrap, see [14]. Just below follow the results stored in the object `results` (output of the `goodness.fit` function) and a plot with the fitted Exp-Weibull density.

```
R> results
$W
[1] 0.07047089

$A
[1] 0.4133608

$KS

One-sample Kolmogorov-Smirnov test

data:  data
D = 0.064568, p-value = 0.7987
alternative hypothesis: two-sided

$mle
[1] 0.3731249 2.4058010 1.3198053

$AIC
[1] 288.6641

$CAIC
[1] 288.9141

$BIC
[1] 296.4796

$HQIC
[1] 291.8272

$Erro
[1] 0.06265212 0.60467076 0.59835491

$Value
[1] 141.332

$Convergence
[1] 0
```

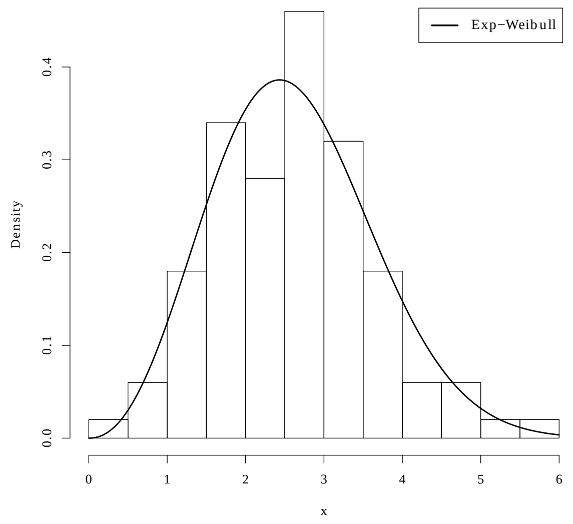

Fig 8. Fitted Exp-Weibull density to stress data (until fracture) of carbon fibers in Gba.

Notes: (*i*) The Kolmogorov-Smirnov statistic may return `NA` with a certain frequency. The return `NA` informs that this statistic is not reliable for the current data set. More details about this issue can be obtained with `help(ks.test)`. In situations where `results$Convergence==0` is `TRUE`, there was convergence for the method passed as an argument to the `method` that minimizes the log-likelihood function multiplied by -1, that is, it minimizes `-log(likelihood)`. (*ii*) The convergence criterion as well as other details about possible values returned by `results$Convergence` can be obtained with `help(optim)` if the argument `method` of the `goodness.fit` function receives the strings `"BFGS"`, `"Nelder-Mead"`, `"SANN"` or `"CG"` (or such those initials letters `"B"`, `"N"`, `"S"` or `"C"`). For the PSO methodology of minimization of the -log(likelihood) function (default `method = "PSO"`), the convergence criterion is displayed as discussed in Section 2, which normally is satisfied. (*iii*) The code for obtaining Figure 8 is presented below:

```
R> pdf(file = "plot_adjustment.pdf", width = 9, height = 9,
+      paper = "special", family = "Bookman", pointsize = 14)

x = seq(0, 6, length.out = 250)
hist(carbone, probability = TRUE, xlab = "x", main = "")
lines(x, pdf_expweibull(par = results$mle, x), lwd = 2)

legend("topright", legend = c(expression(paste("Exp-Weibull"))),
+  lwd = c(2.5), inset = 0.03, lty = c(1), cex = 1.1, col = c("black"))

dev.off()
```

4.2 TTT plot

Several aspects of an absolutely continuous distribution can be seen more clearly from the hazard rate function (hrf) than from either the distribution or density functions.

The hrf is an important quantity characterizing life phenomena. Let X be a random variable with pdf $f(x)$ and cdf $F(x)$. The hrf of X is defined by

$$h(x) = \frac{f(x)}{1 - F(x)},$$

where $1 - F(x)$ is the survival function.
The hrf may be increase, decrease, constant, upside-down bathtub, bathtub-shaped or indicate a more complicated process. In many applications there is a qualitative information about the hazard rate shape, which can help in selecting a specified model. In this context, a device called total time on test (TTT) or its scaled TTT transform proposed by [15] may be used for obtaining the empirical behavior of the hrf. The scaled TTT transform if defined by $(0 < u < 1)$

$$\phi_X(u) = \frac{H_X^{-1}(u)}{H_X^{-1}(1)},$$

where $H_X^{-1}(u) = \int_0^{Q(u)} [1 - F(x)]dx$ and $Q(u)$ is the quantile function of X. The quantity $\phi_X(\cdot)$ can be empirically approximated by

$$T(i/n) = \frac{\sum_{k=1}^{i} X_{k:n} + (n-i)X_{i:n}}{\sum_{k=1}^{n} X_k},$$

where $i = 1, \ldots, n$ and $X_{k:n}$, $k = 1, \ldots, n$, are the order statistics of the sample. Thus, the TTT plot is obtained by plotting $T(i/n)$ against i/n. We can detect the type of hazard rate that the data have. It is a straight diagonal for constant failure rates, it is convex for decreasing failure rates and concave for increasing failure rates. It is first convex and then concave if the failure rate is bathtub-shaped. It is first concave and then convex if the failure rate is upside-down bathtub. For more details, see [15].
The computation of the TTT plot is proposed in the `AdequacyModel` package. The data set named `carbone` will now be used to illustrate the TTT plot function of this package. The real data sets correspond to a data set from [13] on breaking stress of carbon fibres (in Gba). In order to obtain the TTT curve, the `TTT` function has been developed. The following instructions illustrate these functions:

```
R> library(AdequacyModel)
R> data(carbone)
R> TTT(carbone, col = "red", lwd = 2.5, grid = TRUE, lty = 2)
```

The TTT plot for the carbone data set [13] is displayed in Figure 9, which reveals increasing hrf. This plot indicates that distributions with increasing hrf seem to be appropriate for modeling the carbone data set, so that several distributions that have increasing hrf could be good candidates; see the theoretical plot in Figure 1 in [15].

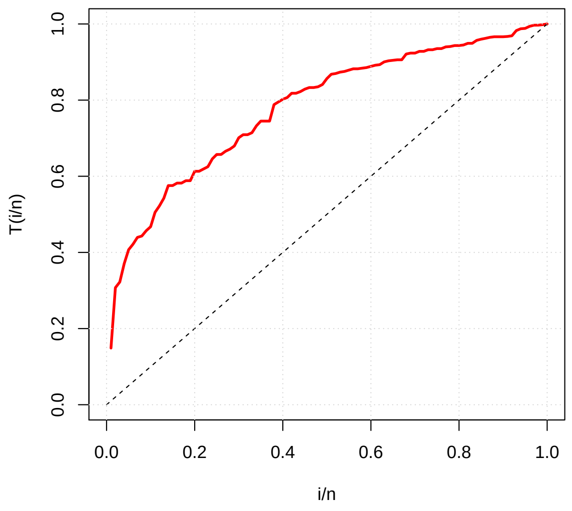

Fig 9. TTT-plot for carbon data.

References

1. Almalki SJ, Nadarajah S. Modifications of the Weibull distribution: a review. Reliability Engineering and System Safety. 2014;124:32–55.

2. Tahir MH, Cordeiro GM. Compounding of distributions: a survey and new generalized classes. Journal of Statistical Distributions and Applications. 2016; p. 3–13.

3. Kennedy J, Eberhart RC. Particle swarm optimization. Proceedings of the 1995 IEEE international conference on neural networks (Perth, Australia). 1995; p. 1942–1948.

4. Kennedy J, Eberhart RC, Shi Y. Swarm intelligence. The Morgan Kaufmann: San Francisco, U.S.A.; 2001.

5. Ying L, Hongli G, Qing W. Adaptive Enhancement with Speckle Reduction for SAR Images Using Mirror-Extended Curvelet and PSO. In: 20th International Conference on Pattern Recognition (ICPR); 2010. p. 4520–4523.

6. Khan A, Bawane NG, Bodkhe S. An Analysis of Particle Swarm Optimization with Data Clustering-Technique for Optimization in Data Mining. International Journal on Computer Science and Engineering. 2010;02(07):2223–2226.

7. Lins ID, Moura MC, Zio E, Droguett EL. A Particle Swarm-optimized Support Vector Machine for Reliability Prediction. Quality and Reliability Engineering International. 2012; p. 141–158.

8. Andras P. A Bayesian Interpretation of the Particle Swarm Optimization and Its Kernel Extension. PLoS One. 2012;7(11).

9. Wang FK, Huang PR. Implementing particle swarm optimization algorithm to estimate the mixture of two Weibull parameters with. Journal of Statistical Computation and Simulation. 2013;84(9):1975–1989.

10. R Core Team. R: A Language and Environment for Statistical Computing; 2017. Available from: https://www.R-project.org/.

11. Bratton D, Blackwell T. A simplified recombinant PSO. Journal of Artificial Evolution and Applications. 2008;2008:14.

12. Chen G, Balakrishnan N. A general purpose approximate goodness-of-fit test. Journal of Quality Technology. 1995;27:154–161.

13. Nichols MD, Paggett W. A bootstrap control chart for Weibull percentiles. Quality and Reliability Engineering International. 2006;22:141–151.

14. Davison A, Hinkley D. Bootstrap methods and their applications. vol. 1. Cambridge University Press; 1997.

15. Aarset MV. How to identify bathtub hazard rate. IEEE Transactions Reliability. 1987;36:106–108.