J. Nonlinear Sci. Appl., 18 (2025), 250–258

ISSN: 2008-1898

Journal of Nonlinear Sciences and Applications

Journal Homepage: www.isr-publications.com/jnsa

Transmuted power function distribution revisited: simulation study and important lessons on starting values and local maxima

Issa Cherif Geraldo

Laboratoire d'Analyse, de Modélisations Mathématiques et Applications (LAMMA), Département de Mathématiques, Faculté des Sciences, Université de Lomé, 1 B.P. 1515 Lomé 1, Togo.

Abstract

In this paper, we consider the transmuted power function distribution (TPFD), an important flexible distribution with applications in lifetime modelling. In past studies, maximum likelihood (ML) was recommended as the best method to use for estimating the parameters of the TPFD. We study the ML estimation problem from a novel computational angle by making a comprehensive study in R software using some of the best optimization algorithms. This study enables us to draw some important lessons on some of the computational aspects sometimes neglected in solving maximum likelihood estimation problems.

Keywords: Numerical optimization, iterative method, maximum likelihood, parameter estimation, probability distributions.

2020 MSC: 60E05, 62F10, 62F30.

1. Introduction

The power function distribution (PFD) is a generalization of the classical uniform distribution and an important distribution in probability and statistics with applications in lifetime modelling [15]. Even if several forms of power function distributions (PFD) are available in the literature, each form is obtainable from the others through a change of variable (see [15] for more details). In order to improve the flexibility of PFD in fitting lifetime data, Shahzad and Asghar [13] considered the PFD with cumulative distribution function (cdf) defined by

$$G(x) = \begin{cases} 0, & \text{if } x \leqslant 0, \\ x^{\alpha}, & \text{if } 0 < x < 1, \\ 1, & \text{if } x \geqslant 1, \end{cases}$$

and applied the quadratic rank transmutation map of Shaw and Buckley [14] to obtain the transmuted power function distribution (TPFD). The cdf and probability density function (pdf) of TPFD are respectively defined for all $0 < x < 1$ by

$$F(x) = \begin{cases} 0, & \text{if } x \leqslant 0, \\ (1+\lambda)x^{\alpha} - \lambda x^{2\alpha}, & \text{if } 0 < x < 1, \\ 1, & \text{if } x \geqslant 1, \end{cases} \tag{1.1}$$

Email address: cherifgera@gmail.com (Issa Cherif Geraldo)

doi: 10.22436/jnsa.018.04.03

Received: 2025-04-23 *Revised:* 2025-05-29 *Accepted:* 2025-06-24

and

$$f(x) = \begin{cases} \alpha x^{\alpha-1}\left(1+\lambda-2\lambda x^{\alpha}\right), & \text{if } 0 < x < 1, \\ 0, & \text{otherwise,} \end{cases}$$

where $\alpha > 0$ and $-1 \leqslant \lambda \leqslant 1$. The TPFD, as defined by equation (1.1), is a special case of the three-parameter TPFD proposed in [1].

Shahzad and Asghar [13] provided several mathematical properties of the TPFD and studied its parameter estimation using the maximum likelihood (ML) method. Tanış [16] provided new mathematical properties of the TPFD that were not examined in [13] and studied its parameter estimation by comparing several methods such as maximum likelihood (ML), least-squares, weighted least-squares, Anderson-Darling and Crámer-Von-Mises methods through simulation study. Of all the compared methods, ML was found to be the best, and Tanış [16] recommended ML as the best method for estimating the TPFD parameters.

In this paper, we study the estimation problem from a novel computational angle. Indeed, estimating the parameters α and λ via the ML method leads to a non-linear optimization problem with box constraints that is analytically intractable, therefore requiring the use of a numerical optimization algorithm. It is well known that there are many optimization algorithms but none of them is perfect, each of them having strengths and weaknesses. Sometimes, algorithms can diverge or produce estimates that are relatively far from the truth and, for the same data, it is even possible that, starting from a same point, some algorithms fail while others succeed [4, 5]. Thus, even though solving the ML estimation problem looks simple (for example, because we only have two parameters to estimate), it poses a computational challenge that deserves to be studied. So, we make a comprehensive study of the ML estimation problem in R software [2] using some of the best optimization algorithms. Our goal is to increase the awareness of scientific researchers to some of the computational aspects sometimes neglected in solving maximum likelihood estimation problems.

The remainder of this paper is organized as follows. In Section 2, we present the ML estimation problem along with the optimization algorithms selected for solving this latter. In Section 3, we present the main results of our simulation-based studies. The paper concludes with Section 4, which contains concluding remarks.

2. Maximum likelihood estimation of the parameters of the TPFD

2.1. Presentation of the problem

Let $x_1, \ldots, x_n$ be n independent observations from the TPFD with unknown parameter vector $\theta = (\alpha, \lambda) \in \mathbb{R}_+^* \times [-1, 1]$. Certainly, there exist several methods for estimating the parameter vector θ. Using simulation study, Tanış [16] compared five methods for estimating θ, namely maximum likelihood (ML), least-squares (LS), weighted LS, Anderson-Darling and Cramer-Von-Mises, and recommended ML because this latter had the lowest mean squared errors (MSE).

The maximum likelihood estimator (MLE) $\hat{\theta} = (\hat{\alpha}, \hat{\lambda})$ of θ is solution to the following constrained non-linear optimization problem:

$$\hat{\theta} = \underset{\theta \in \mathbb{R}_+^* \times [-1,1]}{\arg\max} \ \ell(\theta), \tag{2.1}$$

where

$$\ell(\theta) = n \log(\alpha) + (\alpha - 1) \sum_{i=1}^{n} \log(x_i) + \sum_{i=1}^{n} \log\left(1 + \lambda - 2\lambda x_i^{\alpha}\right),$$

is the log-likelihood function (see [13] for more details). Shahzad and Asghar [13] suggested that $\hat{\theta} = (\hat{\alpha}, \hat{\lambda})$ could be computed as the stationary point of ℓ, i.e., the solution to the following system of equations

$$\nabla \ell(\theta) = \left(\frac{\partial \ell}{\partial \alpha}, \frac{\partial \ell}{\partial \lambda}\right)^{\mathsf{T}} = (0, 0)^{\mathsf{T}}, \tag{2.2}$$

where $\nabla\ell(\theta)$ is the score function (the gradient of the log-likelihood),

$$\frac{\partial\ell}{\partial\alpha} = \frac{n}{\alpha} + \sum_{i=1}^{n}\log(x_i) - \sum_{i=1}^{n}\frac{2\lambda x_i^{\alpha}\log(x_i)}{1+\lambda-2\lambda x_i^{\alpha}} \quad \text{and} \quad \frac{\partial\ell}{\partial\lambda} = \sum_{i=1}^{n}\frac{1-2x_i^{\alpha}}{1+\lambda-2\lambda x_i^{\alpha}}.$$

The function $\ell(\theta)$ integrates the logarithmic barrier through the presence of the term $n\log(\alpha)$ [9]. This latter diverges to $-\infty$ if α tends to 0, thus preventing the case $\alpha \leqslant 0$ from occurring [9]. However, a question that may arise is how the box constraint $-1 \leqslant \lambda \leqslant 1$ is taken into account. Indeed, in the system of equations $\nabla\ell(\theta) = (0,0)^T$, nothing prevents λ from taking values outside the interval $[-1,1]$. We will give an example in Section 3 using simulated samples to prove that looking for the stationary point of $\ell(\theta)$ or solving the system of equations $\nabla\ell(\theta) = (0,0)^T$ does not always lead to the MLE.

To compute the MLE, it is better to solve equation (2.1). Unfortunately, this is a non-linear optimization problem with box constraints that cannot be solved analytically. It is necessary to use a numerical optimization algorithm. A simulation study of the numerical optimization of $\ell(\theta)$ was carried out by Shahzad and Asghar [13]. The latter authors noticed that the estimates obtained by the maximum likelihood method were increasingly close to the true parameters as sample size increased. However, they provided no details regarding the optimization algorithm used in this study. Tanış [16] also considered solving equation (2.1) by using the quasi-Newton Broden-Fletcher-Goldfarb-Shanno (BFGS) algorithm (described later in this paper) and noticed that the MSE defined by

$$\mathrm{MSE}(\hat{\theta}|\theta) = \frac{1}{2}\|\hat{\theta}-\theta\|^2 = \frac{1}{2}\left((\hat{\alpha}-\alpha)^2 + (\hat{\lambda}-\lambda)^2\right),$$

where $\theta = (\alpha,\lambda)$ is the true value of the parameter vector and $\hat{\theta} = (\hat{\alpha},\hat{\lambda})$ is the estimate, decreases with increasing sample sizes.

It is well known that there are many optimization algorithms, and none of them is perfect, each having strengths and weaknesses. Algorithms can sometimes diverge or yield estimates far from the true value and, for the same estimation problem, it is even possible that, starting from the same point, some algorithms fail to find the MLE where others have succeeded [4, 5]. Thus, even though solving Equation (2.1) looks simple, it poses a computational challenge that deserves to be studied.

In this paper, we make a comprehensive numerical study of the numerical optimization problem (2.1) using simulated samples in R software [2]. Taking into account the availability in R software of optimization algorithms that can handle box or inequality constraints, we selected the following three algorithms: (a) the Newton-Raphson algorithm; (b) the quasi-Newton BFGS algorithm; and (c) the derivative-free Nelder-Mead algorithm. These three algorithms are described below.

2.2. *A brief presentation of the selected algorithms*

The following descriptions are based on references such as [3, 5, 7, 9, 10, 12]. This is by no means a detailed review of the properties of the selected optimization algorithms and for more details, we refer the reader to the cited references.

Newton-Raphson (NR) algorithm is the most important and the most applied optimization algorithm [9]. Starting from an initial estimate $\theta^{(0)}$, this algorithm computes successive iterates using the formula

$$\theta^{(k+1)} = \theta^{(k)} - \left[\nabla^2\ell(\theta^{(k)})\right]^{-1}\nabla\ell(\theta^{(k)}),$$

where $\theta^{(k)}$ is the estimate after k iterations ($k \in \mathbb{N}$), $\nabla\ell(\theta^{(k)})$ is the gradient of the log-likelihood evaluated at $\theta^{(k)}$ and

$$\nabla^2\ell(\theta^{(k)}) = \begin{pmatrix} \dfrac{\partial^2\ell}{\partial\alpha^2}(\theta^{(k)}) & \dfrac{\partial^2\ell}{\partial\alpha\partial\lambda}(\theta^{(k)}) \\ \dfrac{\partial^2\ell}{\partial\lambda\partial\alpha}(\theta^{(k)}) & \dfrac{\partial^2\ell}{\partial\lambda^2}(\theta^{(k)}) \end{pmatrix},$$

is the Hessian matrix (the matrix of second partial derivatives) evaluated at $\theta^{(k)}$. The NR algorithm has two interesting properties: (a) it usually converges quickly to the desired unknown solution, say $\hat{\theta}$, if the initial guess $\theta^{(0)}$ is chosen sufficiently close to $\hat{\theta}$ [7]; (b) it also converges quickly in low-dimensional problems [10]. However, it also has weaknesses: (a) its implementation requires computing second partial derivatives and solving linear equations at each iteration in order to inverse the Hessian matrix; (b) it can diverge severely; (c) in maximization problems, there is no guarantee that the next iterate yields a greater log-likelihood value, i.e., the ascent property $\ell(\theta^{(k+1)}) \geqslant \ell(\theta^{(k)})$ is not guaranteed to be satisfied.

As their name suggests, quasi-Newton algorithms are inspired by the NR algorithm but use approximation formulas based on first derivatives to estimate $\left[\nabla^2\ell(\theta^{(k)})\right]^{-1}$ such that, in maximization problems, the ascent property $\ell(\theta^{(k+1)}) \geqslant \ell(\theta^{(k)})$ is satisfied. Among the various possible formulas, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula stands out as the most famous and effective (see chapter 6 of [12] for more details).

The Nelder-Mead (NM) algorithm [3] is one of the most popular and widely used derivative-free algorithms (the term "derivative-free" meaning that no derivative of $\ell(\theta)$ is used during the computation of the iterates). In maximization problems, iterations of the NM algorithm are based on an evolving simplex such that, at each iteration, the vertex with the lowest log-likelihood value is replaced by a new vertex derived from operations on the remaining vertices (see [3] for more details).

3. Simulation-based studies

We study the ML estimation of the parameter vector θ of the TPFD using three high-performing optimization algorithms available in R software [2], as described in Subsection 2.2. The NR algorithm is implemented using the R package **Bhat** [11], while the BFGS and NM algorithms are implemented using the **alabama** package [17]. For all algorithms, the convergence precision is 10^{-6}. In this paper, TPFD samples are generated using the order statistics method suggested in Subsection 1.1 of [6].

3.1. Study 1: convergence of algorithms to values that are not the MLE

Consider the dataset in Table 1, which contains $n = 100$ samples randomly generated from TPFD with $\alpha = 2$ and $\lambda = 0.8$.

Table 1: Example of dataset for which the MLE is not a stationary point.

0.756	0.575	0.518	0.715	0.678	0.386	0.188	0.678	0.382	0.491
0.443	0.447	0.315	0.576	0.574	0.577	0.739	0.926	0.714	0.690
0.682	0.671	0.471	0.333	0.112	0.264	0.109	0.643	0.646	0.719
0.524	0.718	0.635	0.683	0.123	0.509	0.182	0.576	0.517	0.080
0.816	0.891	0.477	0.774	0.246	0.585	0.699	0.289	0.890	0.377
0.360	0.454	0.717	0.219	0.640	0.529	0.271	0.678	0.375	0.481
0.564	0.253	0.144	0.512	0.370	0.498	0.647	0.650	0.837	0.488
0.238	0.496	0.412	0.280	0.450	0.222	0.366	0.332	0.551	0.342
0.729	0.361	0.783	0.641	0.633	0.913	0.586	0.153	0.362	0.600
0.235	0.858	0.628	0.586	0.599	0.943	0.420	0.836	0.448	0.909

To illustrate the potential convergence of algorithms to values that are not the MLE, we ran each algorithm a thousand times, selecting a random starting point in the parameter space $\mathbb{R}_+^* \times [-1, 1]$ for each run. For each algorithm, we computed the convergence proportion (CP), which is the ratio of the number of times the algorithm converged to the total number of executions (one thousand). Table 2 presents the final estimates (rounded to three decimal places) obtained by the selected algorithms.

Table 2: Estimation results for the example dataset (CP stands for convergence proportion).

Algorithm	Final estimates and log-likelihoods		
	$\hat{\theta}_1 = (1.962, 1.000)$ $\ell(\hat{\theta}_1) = 16.430$	$\hat{\theta}_2 = (0.657, -1.000)$ $\ell(\hat{\theta}_2) = 3.401$	
	CP to $\hat{\theta}_1$ (%)	CP to $\hat{\theta}_2$ (%)	Total CP (%)
BFGS	91.4	8.6	100
NM	88.1	11.9	100
NR	17.6	7.5	25.1

From Table 2, we first notice that BFGS and NM algorithms have a convergence proportion (CP) of 100%, while the NR algorithm's CP is only 25.1%. We also notice that the algorithms converged to two distinct values, denoted as $\hat{\theta}_1$ and $\hat{\theta}_2$. However, $\hat{\theta}_1$ is the MLE because it has a higher log-likelihood value. Figure 1, which represents the log-likelihood for the dataset in Table 1, clearly shows two maxima associated with the two estimates, represented by two black dots.

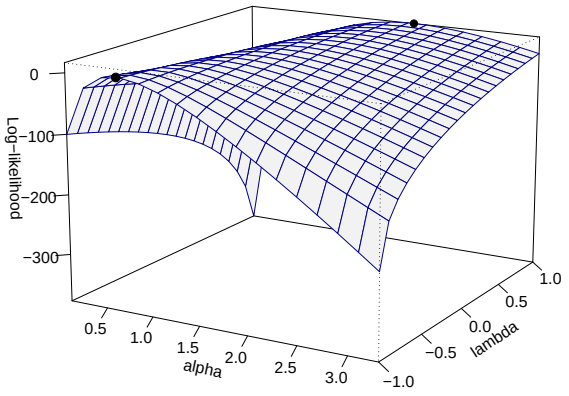

Figure 1: Log-likelihood of the dataset in Table 1.

These results suggest that each algorithm's ability to converge to the MLE depends somewhat on the starting point, though this dependence appears stronger for the NR algorithm. Indeed, NR converged to the MLE for only 17.6% of the thousand starting points, while BFGS and NM converged to the MLE for 91.4% and 88.1%, respectively.

3.2. Study 2: the MLE is not always a stationary point

In Subsection 2.1, we emphasized the fact that computing the stationary point of $\ell(\theta)$, i.e., solving the system of equations $\nabla\ell(\theta) = (0,0)^T$ does not always yield the MLE because nothing prevents λ from taking values outside the interval $[-1,1]$. Rounding to three decimal places, we obtain

$$\nabla\ell(1.962, 1.000) \simeq \begin{pmatrix} -0.007 \\ 8.400 \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

which proves that the MLE is not a solution to the non-linear system of equations (2.2). To compare the solutions of optimization problem (2.1) and non-linear system of equations (2.2), we need an algorithm for solving the system (2.2). There is a variant of the Newton-Raphson algorithm that can be used to numerically solve the later system. In this work, it is implemented using the **nleqslv** package [8]. We noticed that the system (2.2) has several solutions, but only one belongs to the parameter space. Rounded to three decimal places, this solution is $\hat{\theta} = (0.924, -0.499)$ with a log-likelihood of $\ell(\hat{\theta}) = 0.1585$.

This example demonstrates that, to compute the MLE, it is better to solve the optimization problem (2.1) rather than the system of equations (2.2).

3.3. Study 3: influence of the starting point on the performance of the selected algorithms

This third simulation study, whose design is inspired by [4, 5], is based on generating TPFD samples of size $n \in \{50, 200, 1000\}$ from the TPFD with parameters $(\alpha, \lambda) = (2, 0.7)$ and $(\alpha, \lambda) = (1.6, -0.5)$. For each value of (α, λ) and each value of n, we replicated one thousand times, the process of generating a sample of size n and applying the three selected algorithms to estimate the parameters. In addition to parameter estimates, we also computed the convergence proportions and the central process unit (CPU) times for each algorithm. Tables 3 and 4 present the mean values over 1000 replications. For the NR algorithm having a convergence proportion strictly less than 100%, the mean values were calculated only for the successful replications.

Table 3: Estimation results for $(\alpha, \lambda) = (2, 0.7)$ over 1000 replications (standard deviations are in brackets). CP stands for convergence proportions.

		NR	BFGS	NM
$n = 50$	$\hat{\alpha}$	1.494 (0.673)	1.829 (0.521)	1.845 (0.506)
	$\hat{\lambda}$	−0.030 (0.851)	0.440 (0.640)	0.462 (0.623)
	MSE	0.980	0.389	0.362
	CP (%)	24.4	100	100
	CPU time (s)	0.020	0.004	0.009
	Log-likelihood	5.200	5.787	5.833
$n = 200$	$\hat{\alpha}$	1.492 (0.645)	1.838 (0.453)	1.860 (0.428)
	$\hat{\lambda}$	0.002 (0.861)	0.472 (0.592)	0.502 (0.560)
	MSE	0.949	0.316	0.278
	CP (%)	26	100	100
	CPU time (s)	0.035	0.009	0.023
	Log-likelihood	18.441	20.913	21.055
$n = 1000$	$\hat{\alpha}$	1.753 (0.529)	1.809 (0.458)	1.841 (0.427)
	$\hat{\lambda}$	0.362 (0.709)	0.438 (0.616)	0.481 (0.573)
	MSE	0.477	0.347	0.291
	CP (%)	21	100	100
	CPU time (s)	0.084	0.045	0.110
	Log-likelihood	95.920	100.217	101.106

Table 4: Estimation results for $(\alpha, \lambda) = (1.6, -0.5)$ over 1000 replications (standard deviations are in brackets). CP stands for convergence proportions.

		NR	BFGS	NM
$n = 50$	$\hat{\alpha}$	1.750 (0.563)	1.884 (0.576)	1.880 (0.568)
	$\hat{\lambda}$	−0.422 (0.442)	−0.294 (0.443)	−0.296 (0.440)
	MSE	0.270	0.326	0.318
	CP (%)	29.8	100	100
	CPU time (s)	0.024	0.004	0.011
	Log-likelihood	12.591	12.442	12.434
$n = 200$	$\hat{\alpha}$	1.778 (0.565)	1.702 (0.362)	1.682 (0.365)
	$\hat{\lambda}$	−0.368 (0.481)	−0.420 (0.308)	−0.438 (0.314)
	MSE	0.299	0.121	0.121
	CP (%)	31.7	100	100
	CPU time (s)	0.031	0.009	0.024
	Log-likelihood	44.012	46.996	46.996
$n = 1000$	$\hat{\alpha}$	1.977 (0.745)	1.608 (0.228)	1.596 (0.229)
	$\hat{\lambda}$	−0.179 (0.654)	−0.497 (0.205)	−0.508 (0.207)
	MSE	0.613	0.047	0.047
	CP (%)	42.8	100	100
	CPU time (s)	0.071	0.031	0.078
	Log-likelihood	183.181	232.102	232.095

A few important remarks can be made. Firstly, the NR algorithm is very sensitive to the starting

point, and its convergence is not always guaranteed. Indeed, the convergence proportion of the NR algorithm varies between 21% and 42.8%. Secondly, we note that the BFGS algorithm ranks first in terms of computation time. Thirdly, the MSEs are relatively high, especially for $(\alpha, \lambda) = (2, 0.7)$, even for large values of the sample size n. Moreover, MSEs do not always decrease as sample size increases. This indicates that all algorithms converged to a significant extent towards local maxima that are not the maximum likelihood estimates. This further strengthens the conclusion of Subsection 3.1.

3.4. Study 4: improving the performance of selected algorithms via a multiple starting point strategy

Subsection 3.3 makes it clear that for a given dataset, computing the MLE using any of the selected algorithms from a random starting point carries a significant risk of converging to a local maximum that is not the MLE. To reduce this risk and improve algorithm performance, we re-ran the simulations of Subsection 3.3 but this time at replication i ($i = 1, 2, \ldots, 1000$) and for each algorithm, we: (a) selected m random starting points $\theta_{i1}^{(0)}, \ldots, \theta_{im}^{(0)}$ ($m \in \mathbb{N}^*$); (b) executed m times the algorithm considered from the m starting points to obtain m estimates $\hat{\theta}_{i1}, \ldots, \hat{\theta}_{im}$; and (c) chose as MLE, the value $\hat{\theta}_{ij}$ ($j = 1, \ldots, m$) with the maximum log-likelihood, i.e.,

$$\hat{\theta}_i = \underset{\theta \in \{\hat{\theta}_{i1}, \ldots, \hat{\theta}_{im}\}}{\arg\max} \ \ell(\theta).$$

For each replication, an algorithm is considered to have converged if it converged at least once during the m repetitions. Tables 5 and 6 present the mean values over 1000 replications, with $m = 10$ random starting points tried per replication.

Table 5: Estimation results for $(\alpha, \lambda) = (2, 0.7)$ over 1000 replications, with $m = 10$ starting points tried per replication (standard deviations are in brackets). CP stands for convergence proportions.

		NR	BFGS	NM
$n = 50$	$\hat{\alpha}$	1.808 (0.543)	2.004 (0.326)	2.004 (0.326)
	$\hat{\lambda}$	0.407 (0.670)	0.671 (0.310)	0.671 (0.310)
	MSE	0.433	0.102	0.102
	CP (%)	93.8	100	100
	CPU time (s)	1.631	0.039	0.090
	Log-likelihood	5.796	6.140	6.140
$n = 200$	$\hat{\alpha}$	1.840 (0.459)	2.007 (0.155)	2.007 (0.155)
	$\hat{\lambda}$	0.473 (0.601)	0.699 (0.141)	0.699 (0.141)
	MSE	0.325	0.022	0.022
	CP (%)	95.4	100	100
	CPU time (s)	2.333	0.093	0.226
	Log-likelihood	20.898	21.991	21.991
$n = 1000$	$\hat{\alpha}$	1.887 (0.389)	2.005 (0.064)	2.005 (0.064)
	$\hat{\lambda}$	0.540 (0.519)	0.701 (0.056)	0.701 (0.056)
	MSE	0.229	0.004	0.004
	CP (%)	93	100	100
	CPU time (s)	8.066	0.429	1.075
	Log-likelihood	101.495	106.002	106.002

Comparing Tables 3 and 5, and then Tables 4 and 6, we notice an overall improvement in the results. The convergence proportion of NR has improved considerably, although it remains strictly below 100%. Most of the standard deviations of the estimates have decreased, reflecting a weaker dispersion of the estimated values. The CPU times have increased (as expected with $m = 10$ repetitions per replication) but remain reasonable for the BFGS and NM algorithms.

Overall, both BFGS and NM algorithms outperformed NR and achieved fairly close results. They both had a 100% convergence proportion, much smaller MSEs, and lower CPU times than NR. However, we recommend using the BFGS algorithm because it had a smaller overall CPU time than the NM algorithm.

Table 6: Estimation results for $(\alpha, \lambda) = (1.6, -0.5)$ over 1000 replications, with $m = 10$ starting points tried per replication (standard deviations are in brackets). CP stands for convergence proportions.

		NR	BFGS	NM
$n = 50$	$\hat{\alpha}$	1.924 (0.577)	1.983 (0.566)	1.984 (0.565)
	$\hat{\lambda}$	−0.260 (0.432)	−0.206 (0.413)	−0.205 (0.413)
	MSE	0.341	0.362	0.362
	CP (%)	96.7	100	100
	CPU time (s)	0.881	0.044	0.110
	Log-likelihood	12.445	12.471	12.471
$n = 200$	$\hat{\alpha}$	1.725 (0.389)	1.701 (0.364)	1.697 (0.366)
	$\hat{\lambda}$	−0.400 (0.329)	−0.422 (0.307)	−0.426 (0.309)
	MSE	0.143	0.121	0.122
	CP (%)	98	100	100
	CPU time (s)	0.899	0.092	0.235
	Log-likelihood	46.585	47.020	47.020
$n = 1000$	$\hat{\alpha}$	1.612 (0.294)	1.583 (0.231)	1.579 (0.230)
	$\hat{\lambda}$	−0.494 (0.265)	−0.520 (0.209)	−0.524 (0.209)
	MSE	0.078	0.049	0.049
	CP (%)	99.7	100	100
	CPU time (s)	1.336	0.320	0.739
	Log-likelihood	229.624	232.147	232.149

4. Conclusion

In this paper, we studied the transmuted power function distribution (TPFD) proposed by Shahzad and Asghar [13] from a novel computational angle. We were interested in the maximum likelihood (ML) estimation of the parameters of the TPFD. The ML estimation problem is a non-linear optimization problem with box constraints that requires numerical optimization algorithms due to the complex expression of the log-likelihood function. We studied this problem through intensive simulations in R software [2]. Instead of a single algorithm, we considered three of the best existing ones (Newton-Raphson, quasi-Newton BFGS, and Nelder-Mead).

Our studies revealed the existence of local maxima in the log-likelihood away from the maximum likelihood estimator (MLE), thus explaining why, starting from a given point, algorithms may converge to these local maxima or incorrect parameter estimates that are not the MLE. We then proposed a simple strategy consisting of running algorithms from m random starting points ($m \in \mathbb{N}^*$) in the parameter space and keeping the solution having the maximum log-likelihood. This strategy improved the performance of the algorithms and, of the three studied algorithms, the BFGS algorithm appears to be the best because it had a 100% convergence proportion and the smallest mean squared errors and computation times.

It is acknowledged that employing a multiple starting point strategy increased computation times. However, the computation times for the BFGS and Nelder-Mead algorithms remained quite small, and this increase represents a reasonable price to pay to preclude erroneous conclusions regarding the MLE value.

Acknowledgments

The author thanks the reviewers for their careful reading of the paper and for their questions and suggestions which helped improve the paper.

References

[1] M. Ahsan ul Haq, N. S. Butt, R. M. Usman, A. A. Fattah, *Transmuted power function distribution*, Gazi Univ. J. Sci., **29** (2016), 177–185. 1

[2] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, (2024). 1, 2.1, 3, 4
[3] A. Galántai, *The Nelder-Mead simplex algorithm is sixty years old: New convergence results and open questions*, Algorithms, **17** (2024), 21 pages. 2.2, 2.2
[4] I. C. Geraldo, *A study on the estimation of the transmuted generalized uniform distribution*, Int. J. Sci. Res. Math. Stat. Sci., **7** (2020), 33–39. 1, 2.1, 3.3
[5] I. C. Geraldo, *On the maximum likelihood method for the transmuted exponentiated gamma distribution*, Int. J. Math. Stat., **22** (2021), 20–39. 1, 2.1, 2.2, 3.3
[6] D. C. T. Granzotto, F. Louzada, N. Balakrishnan, *Cubic rank transmuted distributions: inferential issues and applications*, J. Stat. Comput. Simul., **87** (2017), 2760–2778. 3
[7] I. Griva, S. G. Nash, A. Sofer, *Linear and nonlinear optimization*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, (2009). 2.2, 2.2
[8] B. Hasselman, *nleqslv: Solve Systems of Nonlinear Equations*, R package version 3.3.5, (2023). 3.2
[9] K. Lange, *Optimization*, Springer, New York, (2013). 2.1, 2.2
[10] K. Lange, E. C. Chi, H. Zhou, *A brief survey of modern optimization for statisticians*, Int. Stat. Rev., **82** (2014), 46–70. 2.2, 2.2
[11] G. Luebeck, R. Meza, *Bhat: General Likelihood Exploration*, R package version 0.9-12, (2022). 3
[12] J. Nocedal, S. J. Wright, *Numerical optimization*, Springer, New York, (2006). 2.2, 2.2
[13] M. N. Shahzad, Z. Asghar, *Transmuted power function distribution: a more flexible distribution*, J. Stat. Manag. Syst., **19** (2016), 519–539. 1, 1, 2.1, 2.1, 4
[14] W. T. Shaw, I. R. C. Buckley, *The alchemy of probability distributions: beyond Gram-Charlier expansions, and a skew-kurtotic-normal distribution from a rank transmutation map*, arXiv preprint arXiv:0901.0434, (2009), 8 pages. 1
[15] M. H. Tahir, M. Alizadeh, M. Mansoor, G. M. Cordeiro, M. Zubair, *The Weibull-power function distribution with applications*, Hacet. J. Math. Stat., **45** (2016), 245–265. 1
[16] C. Tanış, *On transmuted power function distribution: characterization, risk measures, and estimation*, J. New Theory, **34** (2021), 72–81. 1, 2.1, 2.1
[17] R. Varadhan, *alabama: Constrained Nonlinear Optimization*, R package version 2023.1.0, (2023). 3