On the program implementation of a simple Markov homogeneous random search algorithm of an extremum with Cauchy distributions

A S Tikhomirov
Yaroslav-the-Wise Novgorod State University, 41, ul. B. St. Petersburgskaya, Veliky Novgorod, Russian Federation
E-mail: Alexey.Tikhomirov@novsu.ru

Abstract. A program that implements a simple Markov homogeneous random search algorithm of an extremum with Cauchy distributions is presented. This program allows solving a fairly wide class of problems of finding the global extremum of an objective function with a high accuracy.

1. Introduction
Let the objective function \( f: \mathbb{R}^d \rightarrow \mathbb{R} \) take a minimum value at a single point \( x^* \). Let us consider the problem of finding the global minimum point \( x^* \) with a given accuracy \( \varepsilon > 0 \). One way to solve this problem is to use random search algorithms for the extremum of a function (see [1–16]). Such methods have long been successfully used in solving complex optimization problems. Theoretical studies of the convergence rate of some Markov search algorithms are given in [3, 11–14]. This work is a continuation of [11] and is devoted to a computer program that implements one simple but effective algorithm for a homogeneous Markov monotone search for an extremum. The presented computer program complements the program [16], which implements a homogeneous random search algorithm with normal distributions, but using a different probability distribution to obtain new points in the optimization space. The algorithm of this work is based on the use of the Cauchy distribution to obtain points in the optimization space. In [16], a special mixture of normal probability distributions was used. In this paper, we will also use a mixture of Cauchy distributions, and this mixture will be arranged differently. The algorithm for modeling a mixture of this work is simpler. In addition, simulating a Cauchy distribution is easier than simulating a normal probability distribution. Therefore, in general, the presented algorithm is easier to model, and judging by the numerical experiments it is no less efficient than the algorithm [16]. In addition, the description of the algorithm has become easier and more understandable for users of the program.

2. Statement of the problem
We will consider the space \( X = \mathbb{R}^d \) as an optimization space with the metric
\[
\rho_\infty(x, y) = \max_{1 \leq i \leq d} |x_i - y_i|,
\]
where \( x = (x_1, ..., x_d) \) and \( y = (y_1, ..., y_d) \). We denote the closed ball of radius \( r \) centered at \( x \) by \( B_r(x) = \{ y \in \mathbb{R}^d : \rho_\infty(x, y) \leq r \} \). By \( \mu \) we denote the Lebesgue \( d \)-dimensional measure on Borel subsets \( \mathbb{R}^d \). The metric \( \rho_\infty \) corresponds to the structure of the probability distributions used in the work.
We use a homogeneous Markov monotone random search (see [3, 11–14]), described below using the modeling algorithm to find the minimum point. The notation “η ← P(·)” reads as follows: “get the realization of the random vector η with the distribution P”. Operations of the form k ← 1 and ξ ← x denote the usual assignment operations for numbers and points in the optimization space.

Algorithm 1

Step 1. ξ₀ ← x, k ← 1.
Step 2. ηₖ ← P(ξₖ₋₁, ·).
Step 3. If f(ηₖ) ≤ f(ξₖ₋₁), then ξₖ ← ηₖ, otherwise ξₖ ← ξₖ₋₁.
Step 4. If k = N, then stop the algorithm.
Step 5. k ← k + 1 and go to Step 2.

Here x — starting point of search, N — number of search steps, P(x, ·) — Markov transition function (cm. [3, 11–14]).

In the first step of Algorithm 1, random search is initialized. The starting point of the search is point x (operator ξ₀ ← x), and the number of the next search step k becomes equal to unity (operator k ← 1).

At the second step of algorithm 1, we obtain a new “test” point ηₖ in the optimization space. We randomly select a new “test” point using the distribution P(ξₖ₋₁, ·). The distribution P(ξₖ₋₁, ·) depends on the location of the “old” search point ξₖ₋₁. This dependence allows increasing the efficiency of random searches. The transition function P(x, ·) will be called the trial transition function. We will consider a search whose trial transition function does not explicitly depend on step number k. Such a search is called homogeneous. A search whose trial transition functions explicitly depend on the step number k is called inhomogeneous. Of course, due to inhomogeneity, it is possible to increase the efficiency of the search. But such a dependence complicates the choice of search parameters, and the “right” choice of search parameters can be a very difficult task (see, for example, [7]). Therefore, we restrict ourselves to considering a homogeneous search for the sake of ease of use of the search.

In the third step of Algorithm 1, we compare the new test point ηₖ with the old search point ξₖ₋₁. If the new test point ηₖ is not worse than the old search point ξₖ₋₁ (i.e., if the inequality f(ηₖ) ≤ f(ξₖ₋₁) is fulfilled), then the search goes to the new point ηₖ (the operator ξₖ ← ηₖ is executed), otherwise the search remains at the old point (the operator ξₖ ← ξₖ₋₁ is executed).

In the fourth step of Algorithm 1, we check the condition for stopping the search. In this case, a very simple criterion for stopping the search is selected. The search simply performs a predetermined number of steps N, and stops after that.

Let us note that the second, third, fourth and fifth steps of Algorithm 1 are cyclically repeated N times. The first step of Algorithm 1 is performed only once.

We also note that the random search introduced is monotone in the sense that the inequalities f(ξₖ) ≤ f(ξₖ₋₁) are satisfied for all k ≥ 1.

3. Selection of transition functions of the random search

The key choice question for the type of search under study is the choice of the type of trial transition function P(x, ·). When choosing transition functions, two criteria are usually used. Firstly, the search should be quite effective (require not too many steps to solve the problem). In addition, modeling the distribution of P(x, ·) should be quite simple.

The normal probability distribution is used very often, as trial transition functions (see, for example, [5, 7]). In [16], a special mixture of normal probability distributions was used. In this paper, we will use a mixture of Cauchy distributions, and, in addition, this mixture will be arranged differently.

We will describe the used Cauchy distribution. We will first consider the one-dimensional space ℝ. Let x ∈ ℝ. We use the Cauchy distribution, whose density g(y, r, x) is given by the following formula

\[ g(y, r, x) = \frac{r}{\pi r^2 + (x - y)^2} \]  

where y ∈ ℝ and r > 0 — are the parameters of distribution.
We will now consider the $d$-dimensional space $\mathbb{R}^d$. Let $x = (x_1, ..., x_d) \in \mathbb{R}^d$ and $y = (y_1, ..., y_d) \in \mathbb{R}^d$. We will set the density $p(y, r, x)$ we need as follows

$$p(y, r, x) = \prod_{i=1}^{d} g(y_i, r, x_i), \quad (2)$$

where $y = (y_1, ..., y_d)$ and $r > 0$ — are the parameters of distribution, and function $g$ is set by the formula (1).

We denote the probability distribution with density (2) by $T(x, r, \cdot)$. It remains to determine the rule for constructing a mixture of distributions $T(x, r, \cdot)$ to select a trial transition function. We will use the results of work [11].

The trial transition function of the search in question has two main parameters, and an auxiliary one. The two main parameters are the positive numbers $\nu$ and $\Gamma$, for which the inequalities $0 < \nu < \Gamma$ must hold. The parameters $\nu$ and $\Gamma$ determine the range of variation of the values of the parameter $r$ for the distributions $T(x, r, \cdot)$, which form the mixture of distributions used. The main parameters are set by the user of the program when performing a random search. The value $\nu$ can be chosen close to the required accuracy of the solution to the problem when approximated by argument. The value of $\Gamma$ can be chosen close to the assumed accuracy of the initial approximation (the distance from the initial search point to the minimum point).

An auxiliary parameter is the number $Q > 1$. The auxiliary parameter is not set by the user of the program. Put $q = 1/Q$. For $i \geq 0$ we set the standard deviations $r_i = \Gamma q^i$. Denote $n = \min\{i \geq 0 : r_i \leq \nu\}$. It's clear that

$$n = \lfloor d \ln(\Gamma/\nu)/\ln Q \rfloor. \quad (3)$$

We use a mixture with the same distribution probabilities $T(x, r_i, \cdot)$ with parameters $r_0, ..., r_n$ and the center at the current search point as a trial transition function, i.e. put

$$P(x, \cdot) = \frac{1}{n+1} \sum_{i=0}^{n} T(x, r_i, \cdot). \quad (4)$$

The value of the auxiliary parameter $Q$ is equal to the ratio of the volumes of balls $B_{r_i}(x)$, which define areas in the optimization space (4), i.e., $Q = \mu(B_{r_1}(x))/\mu(B_{r_{i+1}}(x))$. The value $Q$ needs to be chosen so that on one side the balls $B_{r_i}(x)$ form a fairly dense “grid” (the value of $Q$ does not greatly exceed 1), and on the other hand, the value $n$ specified by formula (3) should not be too large. We choose the value of $Q$ from heuristic considerations equal to the famous golden ratio, i.e. we set $Q = (\sqrt{5} + 1)/2$. The following search application examples show the resulting values $n$.

4. Estimation of the complexity of the investigated search

Theoretical results on the complexity of the studied search can be obtained using the results of [11]. We use a random search to find the minimum point $x_*$ with the given accuracy $\varepsilon$ (approximation “by argument”). When approximating by argument, we will be interested in getting the search into the ball $B_{\varepsilon}(x_*)$.

We consider an analog of Algorithm 1 from which the condition for stopping the search is removed (i.e., the fourth step of Algorithm 1 is removed) for a theoretical study of the complexity of random search. Such an algorithm generates an infinite sequence of search points $(\xi_k)_{k=0}^{\infty}$. Through

$$\tau_\varepsilon = \min\{k \geq 0 : \xi_k \in B_{\varepsilon}(x_*)\}$$

we can denote the moment the search first hits the $\varepsilon$-neighborhood of the global minimum point. It is clear that for modeling a new test point in the optimization space (modeling the trial transition function $P(x, \cdot)$ given by formula (4)), it is not necessary to calculate the value of the objective function $f$. Thus, at each iteration $\xi_{k-1} \mapsto \xi_k$ of Algorithm 1, exactly one calculation of the objective function takes
place, and the distribution of the random variable \( \tau_e \) gives us quite complete information about the quality of the random search. Indeed, when performing the \( \tau_e \) steps of the search, the values of the objective function \( f \) are calculated \( \tau_e + 1 \) times.

We consider one characteristic of the rate of convergence of a random search. The complexity of a random search is determined through \( E\tau_e \) (mathematical expectation of \( \tau_e \)) and it makes sense the average number of search steps before it reaches the set \( B_\varepsilon(x_\ast) \).

Using the results of [11], one can obtain a theoretical estimate of the complexity of the search under study. Let the objective function \( f \) be non-degenerate (see [11]). We choose the value of \( \nu \) close to the required accuracy of the solution of the problem when approximating by argument (i.e., to the value \( \varepsilon \)). As a value, \( \Gamma \) we will take the upper estimate of the distance from the starting point of the search to the minimum point. (Here, for a more accurate statement of the presented results, it would be necessary to introduce special concepts of [11].) Using the results of [11], we can show that \( E\tau_\varepsilon = O(\ln^2 \varepsilon) \). Thus, from a theoretical point of view, the studied search is fast (in the sense that its complexity has a good order of dependence on \( \varepsilon \)).

5. Simulation of random search
In this section, we describe a modeling algorithm for a homogeneous Markov monotone random search chosen for implementation. At the preliminary stage (performed even before modeling the points in the optimization space), the parameter values \( r_0, \ldots, r_n \) are calculated and stored in the array. To calculate the values \( r_0, \ldots, r_n \) it is enough to perform \( n \) operations of multiplication, and at the same time, the values of the value \( n \) in typical optimization problems turn out to be very small. The following application examples show the resulting values \( n \).

Modeling a new test search point takes place in two stages. At the first stage, one of the values of the parameters \( r_0, \ldots, r_n \), which were calculated in advance and stored in the array, is randomly selected. To do this, we simply simulate a discrete uniform distribution on the set of indices of the array (on the set of integers from 0 to \( n \)) using the Next method of the Random class of the programming language C#.

The presented modeling algorithm is simpler than the algorithm for modeling a random search for work [16].

At the second stage, we simulate the \( d \)-dimensional probability distribution \( T(x, r, \cdot) \) with the chosen parameter value \( r \in \{r_0, \ldots, r_n\} \) and \( x = (x_1, \ldots, x_d) \in \mathbb{R}^d \). We model \( d \) independent random variables having a Cauchy probability distribution with densities \( g(x, r, z_i) \) to model the \( d \)-dimensional probability distribution \( T(x, r, \cdot) \). These random variables make up the coordinates of the new “test” point \( \eta_k \) in the optimization space.

To simulate random variables having a Cauchy distribution with densities \( g(x, r, z_i) \), we use the well-known properties of the Cauchy distribution. Consider a one-dimensional space \( \mathbb{R} \). Let \( x \in \mathbb{R} \). We use the Cauchy distribution, whose density \( g(y, r, x) \) is given by formula (1). It is known that the class of Cauchy distributions is closed under linear transformations. In particular, if a random variable \( \zeta_0 \) has a Cauchy distribution with a density \( g(0,1,x) \), then a random variable \( \zeta_1 = r\zeta_0 + y \) has a Cauchy distribution with a density \( g(y, r, x) \). In the formula \( \zeta_1 = r\zeta_0 + y \), parameter is \( r > 0 \) and point \( y \in \mathbb{R} \). Thus, to simulate a Cauchy distribution with a density \( g(y, r, x) \), it is sufficient for us to be able to simulate a Cauchy distribution with a density \( g(0,1,x) \).

In this paper, we consider two methods for modeling the Cauchy distribution with density \( g(0,1,x) \), this is the inverse function method and the polar method. Let \( \alpha \) is a random variable uniformly distributed over a segment \([0, 1] \). The inverse function method for modeling the Cauchy distribution reduces to using a modeling formula \( \zeta = \tan(\pi(\alpha - 0.5)) \). Note that the modeling formula \( \zeta = \cot(\pi\alpha) \) of the modified inverse function method, which is usually given in the literature, is less convenient for us due to the fact that the Math class of the C# programming language has a tangent function, but no cotangent function.
The polar method for modeling the Cauchy distribution is as follows. It is necessary to obtain a vector \( \chi \) having a uniform distribution in a semicircle \( \{(x, y) : y > 0, x^2 + y^2 < 1\} \) using the selection method from a rectangle \([-1,1] \times [0,1]\) and take the ratio of its ordinate to the abscissa.

Let \( \alpha_1 \) and \( \alpha_2 \) be independent random variables uniformly distributed over the interval \([0, 1]\). We give an algorithm for modeling a random variable \( \zeta \) with a Cauchy distribution with a density \( g(0,1, x) \).

**Algorithm for modeling the Cauchy distribution by the polar method**

Step 1. Receive \( \alpha_1 \) and \( \alpha_2 \). \( \beta \leftarrow 2\alpha_1 - 1, \delta \leftarrow \beta^2 + \alpha_2^2 \).

Step 2. If \( \delta > 1 \), then go to Step 1.

Step 3. \( \zeta \leftarrow \alpha_2 / \beta \).

On average, the polar method requires \( B / \pi \) calls to the random number generator (to obtain the values of \( \alpha_1 \) and \( \alpha_2 \)). In addition, a random variable \( \delta \) is modeled in this method. This \( \delta \) is uniformly distributed over a segment \([0, 1]\) and independent of a random variable \( \zeta \), since when modeling a random search, it is necessary to simulate many random variables with a Cauchy distribution, the value obtained at the current stage will be used at the next modeling stage. Thus, in all cases except the very first one, for modeling the Cauchy distribution on average \( B / \pi - 1 \approx 1.55 \) random number generator calls will be required.

Numerical experiments showed that the inverse function method turned out to be significantly slower than the polar modeling method. This is due to the fact that the tangent function is evaluated rather slowly.

Since the above stages of modeling the probability distribution \( T(x, r, \cdot) \) are quite simple, then, in general, the random search simulation algorithm is simple and effective (with little complexity). In addition, it is very easy to program. The presented modeling algorithm is simpler than the algorithm for modeling a random search for a work [16] and not much more complicated than an algorithm for modeling a simple random search (the so-called “blind search” [3, 5]) which uses uniform distribution in a pre-fixed area of the optimization space.

6. **Description of the program**

The program is written in C# language in the integrated development environment of Visual Studio Professional 2010. Using the old development environment when writing the program allows even users of computers running Windows XP to work with the project. Of course, users of the new Visual Studio 2019 development environment can also work with the proposed project. The Visual Studio development environment can be used for free, and therefore Visual Studio can be a convenient tool for scientific calculations.

The program has a graphical user interface written using Windows Forms. You can download the program at www.novsu.ru/doc/study/tas1 from the “Random search” folder. The program is available both in the form of an executable file, and in the form of a project containing the source code of the program and allowing the user to edit the program at their discretion.

The double numeric type, which provides an accuracy of 15–16 characters is used for calculations in the program. Let us note that this numerical format limits the possible accuracy of the solution to the problem. Arguing somewhat simplistically, we obtain the following conclusions. If the objective function behaves approximately like a quadratic function in the vicinity of the global minimum, then with approximation accuracy with an argument of the order of \( 10^{-8} \) we obtain the approximation accuracy with respect to the value of a function of the order of \( 10^{-16} \). If the minimum value of the objective function belongs to the interval \((1, 10)\), then the double numeric type, which provides an accuracy of 15–16 digits, will not allow calculating the value of the function with an accuracy above \( 10^{-16} \). Thus, the possible accuracy of solving the problem will be of the order of \( 10^{-7} \) for approximation by argument, and of order \( 10^{-14} \) for approximation of the value of the function. Such accuracy, as a rule, is sufficient from a practical point of view. And such accuracy of solving the problem can be obtained by using the random search program in question when solving not too complicated optimization problems. Of course, if the minimum value of the objective function is equal to zero, and the minimum point is also at zero, then the problem can be solved with much higher accuracy.
It is necessary to set the objective function, search parameters and the starting point of the search to use the search. The search results will be the endpoint of the search (the approximating point of the global minimum) and the value of the objective function at the endpoint of the search.

Search parameters and the starting point of the search are easy to set in the main program window.

It is more difficult to set the objective function. The objective function can be set in two ways. Firstly, you can write the function code directly in the program code (in C#). This method is described in more detail in [15]. When writing code that calculates the value of the objective function, as a rule, minimal information about some programming language like C, C++, C#, Java is sufficient.

Secondly, the objective function can be set in the search program itself (without using Visual Studio). To do this, you need to click the “Set Formula” button in the program, select the “Use Formula” item in the dialog box that opens, specify the dimension of the optimization space and write a formula that defines the objective function. The rules for writing the formula are typical for mathematical programs and are given in [15]. The method for setting the objective function is shown in the text field with the words “We use the function code” or “We use the formula”. The calculation of the value of the formula is implemented using the reverse Polish notation. When using the formula, two additional restrictions apply. The maximum allowable dimension of the optimization space does not exceed 200. The maximum allowable number of numbers in the formula does not exceed 500. In the following application examples, calculating the objective function using the formula (instead of writing code in C#) slowed down the search by about 5 times. This is not critical when solving not too complicated optimization problems.

The dimension of the optimization space is specified when setting the objective function.

In the program, you can set output formats for the value of the objective function and the coordinates of points (see [15]). You can write comments on the task at hand. Comments are written in text format and saved in a file along with parameters and search results.

The program uses a pseudo-random number generator to perform the search. It can be initialized either with a value depending on the system time of the computer, or with a set value.

The program saves data in XML format, and can export key search characteristics in text format.

When setting the value of the “Number of search steps $N$” parameter to zero, the program will calculate the value of the function at the starting point of the search. This can be used to calculate the value of the objective function at a given point.

7. Selecting search parameters

It is important to note that the choice of search parameters can have a major impact on the effectiveness of the random search method [3, 5, 7]. Moreover, many search algorithms contain a large number of heuristic parameters, and it is very difficult for the user of such an algorithm to find “good” parameter values that are suitable for the optimized function. Here is a quote from [7] related to the “very fast annealing method” proposed by L. Ingber: “Among the drawbacks of this method is that it sometimes takes several months to fine-tune it to solve a specific problem because a large number of parameters”. Moreover, with the right selection of parameters, the “very fast annealing method” can show very good results [6, 7].

The presented search has only three parameters, that is the numbers $\nu$, $\Gamma$ and $N$. Positive numbers $\nu$ and $\Gamma$ determine the range of variation of the parameters $r_0, \ldots, r_n$ of the distributions $T(x, r_i, \cdot)$, which form the mixture (4). The inequalities $0 < \nu < \Gamma$ must be held for $\nu$ and $\Gamma$. The third parameter is $N$ — the number of search steps.

The value of $\nu$ can be chosen close to the required accuracy of the solution to the problem when approximated by argument. The value of $\Gamma$ can be chosen close either to the assumed accuracy of the initial approximation (the distance from the initial search point to the minimum point) or to the diameter of the region under study in the optimization space. When choosing $\Gamma$, one can use the upper bound.

It is desirable to take the number of search steps $N$ large enough. When solving a single task, you can, for example, complete a billion search steps, even if the task is simple enough and can be solved
much faster. Modern personal computers may well perform such volumes of computations, at least for not too complex objective functions.

In addition to the three search parameters, it is necessary to select the starting point of the search. It is clear that the starting point is better located closer to the point of global extremum.

The proposed search algorithm is largely free from insurmountable difficulties in choosing parameters. In particular, in the numerical examples of the next section, the minimal tuning of parameters was performed, consisting of literally several attempts to run the program with different parameter values.

8. Examples of using the program
Here are some examples of using the presented program to solve optimization problems. A personal computer with a processor Intel Core i5-4460S was used for calculations.

8.1. Example 1
Let us use the example from [5]. The optimization space is $X = \mathbb{R}^2$, $x = (x_1, x_2)$,

$$f(x) = f(x_1, x_2) = x_1^4 + x_2^4 + x_1 x_2 + x_2^2.$$ 

The function $f$ takes a minimum value at a single point $x_0 = (0, 0)$ and $f(x_0) = 0$. The starting point of the search is $x = (1, 1)$ and $f(x) = 4$. The number of search steps $N$ is $10^4$ here.

Algorithm B of the book [5] obtains the minimum value of the objective function $2.7 \times 10^{-6}$. Algorithm B corresponds to the search for Algorithm 1 using the normal probability distribution as a transition function.

Algorithm C of the book [5] obtains the minimum value of the objective function $2.5 \times 10^{-7}$. Algorithm C also uses the normal probability distribution as a transition function, but is a more complex search option, in which, when constructing a new search point, the displacement made in the previous step of the algorithm is taken into account.

The homogeneous search article algorithm [16] obtains the minimum value of the objective function $7.8 \times 10^{-50}$.

Algorithm 1 of this paper with the parameters $\nu = 2 \times 10^{-23}$ and $\Gamma = 1$ receives the minimum value of the objective function $1.4 \times 10^{-48}$. The value of $n = 218$.

In this example, the search for Algorithm 1 turned out to be much more accurate than the algorithms B and C of the book [5], using the normal probability distribution. Algorithm 1 results are similar to article search results [16].

Algorithm 1 of this work with the parameters $\nu = 10^{-165}$, $\Gamma = 1$, and $N = 10^6$ obtains the minimum value of the objective function equal to zero (i.e., less than the value $5 \times 10^{-324}$, defining a range of values of the type double of the C# programming language) and the minimum point $(-7.2 \times 10^{-163}, -8.1 \times 10^{-163})$. Let us note that in this case the maximum accuracy with which calculations in C# can be performed using the number format double (due to the fact that it is impossible to more accurately calculate the value of the objective function). The search execution time was 0.078 seconds. Using the formula, the search execution time was 0.31 seconds. Using the formula slowed down the search by 4 times. The value of $n = 1580$.

Separately, we should note that when modeling the Cauchy distribution by the inverse function method, the search execution time was 0.14 seconds. The inverse function method slowed down the search by 1.8. Therefore, now (until a quick method for calculating the tangent function appears) the Cauchy distribution by the inverse function method is better not to model.

The search algorithm for article [16] also required $10^6$ steps to obtain a value of the objective function equal to zero. Search execution time was 0.109 seconds. Here, the results of Algorithm 1 are similar to the search results of [16], but Algorithm 1 is faster (due to simplicity of modeling).

8.2. Example 2
The optimization space here is $X = [-8, 8]^2$, $x = (x_1, x_2)$,
\[ f(x) = f(x_1, x_2) = \frac{1}{2} \left( (x_1^4 - 16x_1^2 + 5x_1) + (x_2^4 - 16x_2^2 + 5x_2) \right). \]

The function \( f \) has four local minima, one of which is global. The starting point of the search is \( x = (4.0, 6.4) \) and \( f(x) = 537.18 \). The search for Algorithm 1 with parameters \( v = 10^{-7}, \Gamma = 10, \) and \( N = 20000 \) finds the minimum value of the objective function \(-78.3323314075428\) and the minimum point \((-2.903534, -2.903534)\). The value of \( n \) is 77. Let us note that the extreme accuracy has been achieved with which you can perform calculations in C# using the double number format (due to the fact that it is impossible to calculate the value of the objective function more accurately). The results are close to the results of a homogeneous article search [16].

8.3. Example 3

The space here is \( X = [-4, 4]^3 \), \( x = (x_1, x_2, ..., x_{10}) \).

\[ f(x) = f(x_1, x_2, ..., x_{10}) = \sum_{n=1}^{5} (100(x_{2n} - x_{2n-1})^2 + (1 - x_{2n-1})^2). \]

The function \( f \) is a well-known Rosenbrock test function used for local optimization methods. The function \( f \) takes the minimum value \( f(x_1) = 0 \) at the point \( x_1 = (1, 1, ..., 1) \). The starting point of the search is \( x = (-1.2, 1, -1.2, 1, ..., 1) \) and \( f(x) = 121 \). The search for algorithm 1 with parameters \( v = 10^{-17}, \Gamma = 4, \) and \( N = 10^7 \) finds the minimum value of the objective function \( 5.7 \times 10^{-29} \). The value of \( n \) is 843. The search time was 2.8 seconds. Using the formula, the search time was 14.8 seconds. Using the formula slowed down the search by 5.2 times.

Separately, we note that when modeling the Cauchy distribution by the inverse function method, the search execution time was 6.23 seconds. The inverse function method slowed down the search by 2.2 times.

Here, the results of Algorithm 1 are similar to the search results of [16], but Algorithm 1 is faster. There, the search execution time was 3.5 seconds.

8.4. Example 4

Consider an example with a very simple objective function, but in a very large dimension optimization space for random search methods. Here the space \( X = \mathbb{R}^{1000}, (x_1, x_2, ..., x_{1000}) \), \( f(x) = \sum_{n=1}^{1000} x_n^2 \). The function \( f \) takes the minimum value \( f(x) = 0 \) at a single point. The starting point of the search is \( x = (1, 1, ..., 1) \). The search for Algorithm 1 with parameters \( v = 10^{-11}, \Gamma = 10, \) and \( N = 10^6 \) finds the minimum value of the objective function \( 8.5 \times 10^{-13} \). Value \( n = 57420 \). Search time was 25 seconds.

Separately, we note that when modeling the Cauchy distribution by the inverse function method, the search execution time was 59 seconds. The inverse function method slowed down the search by 2.3 times.

A homogeneous search for the article [16] at \( N = 10^6 \) received the minimum value of the objective function \( 2.3 \times 10^{-14} \). The search execution time was a 31 second. Here, the results of Algorithm 1 are similar to the search results of [16], but Algorithm 1 is faster.

9. Conclusion

The results show that the presented homogeneous search algorithm based on the use of Cauchy distributions is quite effective. The presented random search program can be successfully used to solve optimization problems. The program itself is easy to use, and selecting search parameters is not a difficult task. Moreover, the program allows solving problems with the utmost precision that can be obtained using the double number format of the C# programming language.

References

[1] Ermakov S M and Zhigljavsky A A 1983 On the random search for a global extreme Probability Theory and its Applications 1 129–136
[2] Ermakov S M, Zhigljavsky A A and Kondratovich M V 1989 Comparison of some procedures for random search of a global extremum *Journal of computational mathematics and mathematical physics* Vol. 29 2 163–170

[3] Zhigljavsky A and Zilinskas A 2008 *Stochastic global optimization* (Berlin: Springer-Verlag)

[4] Zhigljavsky A and Zilinskas A 2016 Stochastic global optimization: a review on the occasion of 25 years of Informatica *Informatica* Vol. 27 2 229–256

[5] Spall J C 2003 *Introduction to stochastic search and optimization: estimation, simulation, and control* (New Jersey: Wiley)

[6] Ingber L 1989 Very fast simulated re-annealing *Mathl. Comput. Modelling.* Vol. 12 967–973

[7] Lopatin A S 2005 Annealing method *Stochastic optimization in computer science* 1 133–149

[8] Granichin O N and Polyak B T 2003 Randomized estimation and optimization algorithms with almost arbitrary noise (Moscow: Science)

[9] Sushkov Yu A 1972 On one method of organizing a random search, *Research operations and statistics modeling* (Leningrad: Publishing House of Leningrad State University) 1 180–186

[10] Tarlowski D 2017 On the convergence rate issues of general Markov search for global minimum *Journal of Global Optimization* Vol. 69 4 869–888

[11] Nekrutkin V V and Tikhomirov A S 1993 Speed of convergence as a function of given accuracy for random search methods *Acta Applicandae Mathematicae* 33 89–108

[12] Tikhomirov A S 2006 On the Markov homogeneous optimization method *Computational Mathematics and Mathematical Physics* Vol. 46 3 361–375

[13] Tikhomirov A S 2007 On the convergence rate of the Markov homogeneous monotone optimization method *Computational Mathematics and Mathematical Physics* Vol. 47 5 780–790

[14] Tikhomirov A, Stojunina T and Nekrutkin V 2007 Monotonous random search on a torus: integral upper bounds for the complexity *Journal of Statistical Planning and Inference* Vol. 137 12 4031–4047

[15] Tikhomirov A S 2018 On the program implementation of a Markov homogeneous monotonous random search algorithm of an extremum *IOP Conference Series: Materials Science and Engineering* Vol. 441 012055 1–8

[16] Tikhomirov A S 2019 On the program implementation of a Markov homogeneous random search algorithm of an extremum with normal distributions *Journal of Physics: Conference Series* 1352 012052 1–7