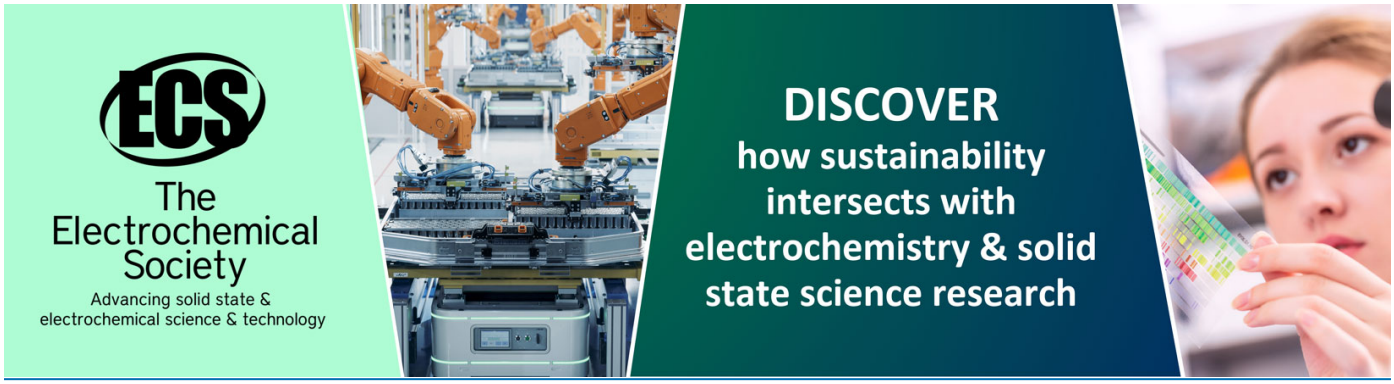
On the program implementation of a Markov inhomogeneous random search algorithm with normal distributions



IOP Conf. Series: Journal of Physics: Conf. Series **1352** (2019) 012053 doi:10.1088/1742-6596/1352/1/012053

On the program implementation of a Markov inhomogeneous random search algorithm with normal distributions

A S Tikhomirov

Yaroslav-the-Wise Novgorod State University, ul. B. St. Petersburgskaya, 41
173003 Veliky Novgorod, Russia

E-mail: Alexey.Tikhomirov@novsu.ru

Abstract. A program that implements a Markov inhomogeneous monotonous random search algorithm of an extremum with normal distributions is presented. This program allows to solve 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 $f\colon \mathbb{R}^d \mapsto \mathbb{R}$ take the minimum value at a single point x_*. Let us consider the task of finding a global minimum point x_* with a given accuracy $\varepsilon > 0$. One way to solve this problem is to use algorithms for randomly searching for the extremum of a function (see [1–18]). Such methods have long been successfully used in solving complex optimization problems. Theoretical studies of the rate of convergence of some Markov search algorithms are given in [3, 11–17]. This paper is a continuation of [11, 17] and is devoted to a computer program that implements one of the algorithms of the inhomogeneous Markov monotone search for an extremum using the normal probability distribution. The presented computer program supplements the program [19], which implements a homogeneous random search algorithm using a different (non-normal) probability distribution.

2. Formulation of the problem

As an optimization space, we will consider the space $X = \mathbb{R}^d$ with the Euclidean metric

$$\rho(x, y) = \left(\sum_{n=1}^{d} (x_n - y_n)^2 \right)^{1/2},$$

where $x = (x_1, \dots, x_d)$ and $y = (y_1, \dots, y_d)$. A closed ball of radius r with center at x is denoted by $B_r(x) = \{y \in \mathbb{R}^d : \rho(x, y) \le r\}$. Let $\mathcal{F}$ be the σ-algebra of Borel sets в $\mathbb{R}^d$. By μ we denote Lebesgue measure on Borel subsets of $\mathbb{R}^d$.

To search for the minimum point, we use the inhomogeneous Markov monotonic random search (see [3, 11, 17]), described later using a simulation algorithm. The notation "$\eta \leftarrow P(\,\cdot\,)$" reads like this: "get the realization of a random vector η with distribution P". For numbers and points in an optimization space, operations of the form $k \leftarrow 1$ and $\xi \leftarrow x$ denote ordinary assignment operations.

Algorithm 1

Step 1. $\xi_0 \leftarrow x$, $k \leftarrow 1$.

Step 2. $\eta_k \leftarrow P_k(\xi_{k-1}, \cdot\,)$.

Step 3. If $f(\eta_k) \le f(\xi_{k-1})$, then $\xi_k \leftarrow \eta_k$, otherwise $\xi_k \leftarrow \xi_{k-1}$.

Step 4. If $k = N$, complete the operation of the algorithm.

Step 5. $k \leftarrow k + 1$ and go to step 2.

Here x — is the starting point of the search, N is the number of search steps, $P_k(x, \cdot)$ are Markov transition functions (see [3, 11, 17]).

In the first step of algorithm 1, the random search is initialized. The starting point of the search becomes the point x (operator $\xi_0 \leftarrow x$), and the number of the next search step k becomes equal to one (operator $k \leftarrow 1$).

At the second step of algorithm 1, we obtain a new "trial" point η_k in the optimization space. We randomly select a new "trial" point using the distribution $P_k(\xi_{k-1}, \cdot)$. The distribution $P_k(\xi_{k-1}, \cdot)$ depends on the search step k and the position of the old search point ξ_{k-1}. Such a relationship can improve the efficiency of random search. The transition functions $P_k(x, \cdot)$ will be called trial transition functions. In [18], a search was considered, the trial transition functions of which were not explicitly dependent on the step number k (that is, they had the form $P_k(x, \cdot) = P(x, \cdot)$). Such a search is called homogeneous. Here we will consider a inhomogeneous search, the trial transition functions $P_k(x, \cdot)$ of which explicitly depend on the step number k. Due to the inhomogeneity can improve search efficiency. But such a dependence complicates the choice of search parameters, and the "right" choice of search parameters can be a difficult task (see, for example, [7]).

In the third step of algorithm 1, we compare the new trial point η_k with the old search point ξ_{k-1}. If the new trial point η_k is not worse than the old search point ξ_{k-1} (that is, if the following inequality holds $f(\eta_k) \le f(\xi_{k-1})$), then the search goes to the new point η_k (the following operator is executed $\xi_k \leftarrow \eta_k$), otherwise the search remains at the old point (the following operator is executed $\xi_k \leftarrow \xi_{k-1}$).

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

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

Note also that the random search introduced is monotonic, in the sense that the inequalities $f(\xi_k) \le f(\xi_{k-1})$ are satisfied for all $k \ge 1$.

3. Selection of trial transition functions

The key question of choice for the type of search under investigation is the choice of the type of trial transition functions $P_k(x, \cdot)$. When choosing transition functions, two criteria are usually used. First, the search must be sufficiently effective (require not too many steps to solve the problem). In addition, the modeling of the $P_k(x, \cdot)$ distributions should be fairly simple.

A homogeneous search was considered in [18], the trial transition functions of which did not explicitly depend on the step number k (that is, they had the form $P_k(x, \cdot) = P(x, \cdot)$). Therefore, there we had to choose one transition function. Here we need to select N different transition functions, where N is the number of search steps. It is clear that this significantly complicates the task. In addition, estimates of labor intensity were obtained and investigated for homogeneous search in [12–14]. These results were used to select a trial homogeneous search transition function. In [18], the transition function $P(x, \cdot)$ was used as a trial transition function of a homogeneous search, which minimizes the estimate of the complexity of a random search when optimizing the simplest objective function. Unfortunately, there are no such general theoretical results for inhomogeneous search.

Very often, the normal probability distribution is used as trial transition functions (see, for example, [5, 7]). In this paper, we also use the normal probability distributions. To construct the search, it remains for us to choose the standard deviations for the normal distributions used. To select these standard deviations we use heuristic considerations. Consider a homogeneous Markov monotone random search, the trial transition function $P(x, \cdot)$ of which minimizes the estimate of the complexity of the random search studied in [12–14] when optimizing the simplest objective function. This search has several advantages. This search (for non-degenerate objective functions) provides a good order of dependence of the obtained estimates of labor intensity on ε (see [3, 12–14]). And the examples of using the program

[19] given in [18] show acceptable efficiency when optimizing not too complex objective functions. A trial transition function $P(x, \cdot)$ of such a homogeneous search is close to a mixture with equal probabilities of uniform distributions in balls whose radii form a geometric progression. The radii of this geometric progression run through the values from the assumed accuracy of the initial approximation (the distance from the starting point of the search to the minimum point) to the required accuracy of solving the problem when approximating with an argument. Therefore, in the inhomogeneous search under study, we will use normal probability distributions, the standard deviations of which form a geometric progression. Unlike homogeneous search, in an inhomogeneous search we will not use a mixture of distributions, but will use these distributions sequentially. We begin naturally with the largest standard deviation, and end with the smallest.

In addition, since the number of search steps must be large enough, in order not to change standard deviations too often, we will break the entire search into stages, and we will change the standard deviation only after the next stage is completed. Thus, trial transition functions will be constant for all steps of a single stage.

By virtue of the foregoing, in this section, we choose for realization a inhomogeneous Markov monotone random search, the trial transition functions of which are the normal probability distributions. The entire search is divided into stages, and the trial transition functions of the search will be constant for all the steps of a single stage. We shall change the standard deviations of the normal distributions only after the next search stage is completed, and these standard deviations form a geometric progression.

4. Simulation of random search

In this section, we present an algorithm for modeling a inhomogeneous Markov monotone random search chosen for realization. Presented search has only four parameters. The parameters defining the range of changes in the standard deviations of normal distributions are the positive numbers ν and Γ, for which the inequalities $0 < \nu \leq \Gamma$ must be satisfied. The third parameter is N — the number of search steps. The fourth parameter is m — the number of steps at the search stage (trial transition functions are constant for all steps of one stage). The parameters m and N are natural numbers satisfying the inequalities $1 \leq m \leq N$.

Let us calculate two auxiliary parameters. The number of stages of the search is $\tau = [N/m]$. The standard deviations at the stages of the search form a geometric progression with the denominator of the progression $q \in (0,1]$, where

$$q = \begin{cases} 1, & \text{for } \tau = 1, \\ (\nu/\Gamma)^{1/(\tau-1)}, & \text{for } \tau > 1. \end{cases}$$

Let $N(x, \sigma, \cdot)$ denote the distribution of a d-dimensional Gaussian random vector with center at $x \in \mathbb{R}^d$ and independent components with the same standard deviation σ.

Now we write down the algorithm for modeling the chosen for the realization of a inhomogeneous Markov monotone random search $\xi_0, \dots, \xi_N$.

Algorithm 2

Step 1. $\xi_0 \leftarrow x$, $k \leftarrow 1$, $\sigma \leftarrow \Gamma$.

Step 2. $\eta_k \leftarrow N(\xi_{k-1}, \sigma, \cdot)$.

Step 3. If $f(\eta_k) \leq f(\xi_{k-1})$, then $\xi_k \leftarrow \eta_k$, otherwise $\xi_k \leftarrow \xi_{k-1}$.

Step 4. If $k = N$, complete the operation of the algorithm.

Step 5. If $k \bmod m = 0$, then $\sigma \leftarrow \sigma * q$.

Step 6. $k \leftarrow k + 1$ and go to step 2.

Here x is the starting point of the search, k is the number of the search step, N is the number of search steps. The standard deviation σ changes after the completion of the each stage of the search, where m is the number of steps in the search stage, and $k \bmod m$ denotes the remainder of k divided by m.

At the second step of Algorithm 2, we obtain a new "trial" point η_k in the optimization space using the Gaussian distribution $N(\xi_{k-1}, \sigma, \cdot)$.

In the third step of algorithm 2, we compare the new trial point η_k with the old search point ξ_{k-1}. If the new trial point η_k is not worse than the old search point ξ_{k-1} (that is, if the inequality $f(\eta_k) \leq f(\xi_{k-1})$) holds, then the search goes to the new point η_k (the operator $\xi_k \leftarrow \eta_k$ is executed), otherwise the search remains at the old point (the operator $\xi_k \leftarrow \xi_{k-1}$ is executed).

In the fourth step of algorithm 2, we check the condition for stopping the search. In this case, the search stops after performing a predetermined number of steps N, i.e. under condition $k = N$. If the search continues (that is, if $k < N$), then at the fifth and sixth steps of algorithm 2 we recalculate the values of the search parameters, and we return to the second step of algorithm 2.

To simulate a d-dimensional Gaussian random vector in the second step of algorithm 2, we simulate d independent normal random variables with zero mean and standard deviation σ. These random variables are added to the coordinates of the current search point ξ_{k-1} to get a new trial point η_k in the optimization space.

To simulate normal random variables, we use the modified polar method. The modified polar method is designed to simulate two independent standard normal random variables ζ_1 and ζ_2. Let α_1 and α_2 be independent uniformly distributed variables on the interval $[0, 1]$.

Standard Normal Distribution Modeling Algorithm

Step 1. Get α_1 and α_2. $\beta_1 \leftarrow 2\alpha_1 - 1$, $\beta_2 \leftarrow 2\alpha_2 - 1$, $\delta \leftarrow \beta_1^2 + \beta_2^2$.

Step 2. If $\delta > 1$, then proceed to step 1.

Step 3. $t \leftarrow \sqrt{-2\ln(\delta)/\delta}$, $\zeta_1 \leftarrow \beta_1 t$, $\zeta_2 \leftarrow \beta_2 t$.

Since the simulation algorithms are quite simple, the random search simulation algorithm is generally fairly easy to program. The simulation algorithm presented is a bit more complicated than the random search simulation algorithm [18] and the simplest random search simulation algorithm (the so-called "blind search" [3, 5]) which uses a uniform distribution in a pre-fixed area of optimization space.

5. Program Description

The program is written in C# in the Microsoft Visual Studio Professional 2010 integrated development environment. 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 also available as an executable file and as a project containing the source code of the program and allowing the user to edit the program at his discretion.

To run the executable file of the program requires the Microsoft.NET Framework 4. Usually it is already installed on the computer, but if necessary it can be downloaded from the Microsoft website. To edit the project you need to install the development environment Microsoft Visual Studio. This development environment can be used for free, and, therefore, this development environment can serve as a convenient tool for scientific calculations.

For calculations, the program uses the numeric type of double, ensuring accuracy of 15–16 characters. Note that this number format limits the possible accuracy of solving the problem. Arguing somewhat simplistic, we get the following conclusions. If the objective function behaves approximately as a quadratic function in the neighborhood of the global minimum, then with an accuracy of approximation with an argument of the order of 10^{-8} we obtain an accuracy of approximation with a value of the function of about 10^{-16}. If the minimum value of the objective function belongs to the interval (1, 10), then the numeric type double, providing accuracy of 15–16 characters, will not allow the function value to be calculated with an accuracy higher than 10^{-16}. Thus, the typical accuracy of the solution of the problem will be about 10^{-7} for the approximation of the argument, and about 10^{-14} for the approximation of the value of the function. Such accuracy is usually sufficient from a practical point of view. And such accuracy of the solution of 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 zero, and the minimum point is also zero, then the problem can be solved with much higher accuracy.

To apply the search, you must specify the objective function, search parameters and the starting point of the search. The search results will be the end point of the search (approximating the global minimum point) and the value of the objective function at the end point 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 defined in two ways. First, you can write the function code directly in the program code (in C#). This method is described in more detail in [18]. When writing code that calculates the value of an objective function, as a rule, only minimal information about a programming language such as C, C++, C#, Java is sufficient.

Secondly, the target function can be set in the search program itself (without using Microsoft Visual Studio). To do this, 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 the formula defining the objective function. The method of specifying the objective function is shown in the text box with the words "Use the function code" or "Use the formula".

The dimension of the optimization space is specified when defining the function.

In the program, you can set the output formats for the value of the objective function and the coordinates of points (see [18]).

You can write comments to the problem being solved. Comments are written in text format and saved in a file along with parameters and search results.

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

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

When you set the value of the parameter "Number of search steps N" 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.

Note that the use of the old Microsoft Visual Studio 2010 development environment when writing a program allows you to work with the project even for users of computers running the Windows XP operating system.

6. The choice of 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]. At the same time, 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 fit the function being optimized. Let us quote from [7], referring to the method of very fast annealing proposed by L. Ingber: "Among the drawbacks of this method is the fact that due to a large number of parameters it sometimes takes several months to properly configure it to solve a specific problem." Moreover, with proper selection of parameters, the method of very fast annealing can show good results [6, 7].

Presented search has only four parameters. The parameters defining the range of changes in the standard deviations of normal distributions are the positive numbers ν and Γ, for which the inequalities $0 < \nu \leq \Gamma$ must be satisfied. The third parameter is N – the number of search steps. The fourth parameter is m – the number of steps at the search stage (trial transition functions are constant for all steps of a single stage).

The value of ν can be chosen close to the required accuracy of the solution of the problem in the approximation of the argument. The value of Γ can be chosen close to either the expected accuracy of the initial approximation (the distance from the starting point of the search to the minimum point) or the diameter of the area under study in the optimization space. When choosing Γ, we can use the upper bound.

The number of search steps N is desirable to take quite large. When solving a single problem, you can, for example, perform a billion search steps, even if the task is simple enough and it can be solved much faster. Modern personal computers can easily perform similar volumes of calculations, at least for

not too complex objective functions. However, for such volumes of calculations, the code of the objective function must be set programmatically in the C# programming language.

The parameter m (number of steps at the search stage) can be set so that it is not necessary to change the standard deviations of normal distributions too often. In the numerical examples in the next section, we used the values $m = 10$ and $m = 100$.

In addition to the four search options, you need 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 the insurmountable difficulties of choosing parameters. In particular, in the numerical examples of the next section, a minimal selection of parameters was carried out, consisting literally of several attempts to launch a program with different values of parameters.

7. Examples of the use of the program

Let us give some examples of using the presented program for solving optimization problems. For calculations, a personal computer with an Intel Core i5-4460S processor was used.

7.1. Example 1

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

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

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

Algorithm B of the book [5] gets the minimum value of the objective function 2.7×10^{-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] gets the minimum value of the objective function 2.5×10^{-7}. Algorithm C also uses the normal probability distribution as a transition function, but represents a more complex search variant, in which the movement made in the previous step of the algorithm is taken into account when constructing a new search point.

Algorithm 1 of homogeneous search [18] gets the minimum value of the objective function 9.9×10^{-49}.

Algorithm 2 in this paper with the parameters $\nu = 10^{-165}$, $\Gamma = 1$ and $m = 10$ gets the minimum value of the objective function equal to zero (i.e., less than the value 5×10^{-324}, which determines the range of values of type double C# programming language) and the minimum point $(8.5 \times 10^{-163}, 4.4 \times 10^{-163})$. Note that in this case, the maximum accuracy is achieved with which you can perform calculations on C# using the double number format (because that it is impossible to more accurately calculate the value of the objective function).

Algorithm 1 of homogeneous search [18] to obtain the zero value of the objective function was required to perform 10^6 steps.

In this example, the search for algorithm 2 turned out to be much more accurate than the algorithms B and C of the book [5] and algorithm 1 of the article [18].

7.2. Example 2

Here is the optimization space $X = [-8, 8]^2$, $x = (x_1, x_2)$,

$$f(x) = f(x_1, x_2) = \frac{1}{2}\Big((x_1^4 - 16x_1^2 + 5x_1) + (x_2^4 - 16x_2^2 + 5x_2)\Big).$$

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 2 with the parameters $\nu = 10^{-8}$, $\Gamma = 10$, $m = 10$ and $N = 20000$ finds the minimum value of the objective function -78.3323314075428 and the minimum point $(-2.903534, -2.903534)$. Note that the maximum accuracy has been reached here,

with which you can perform calculations on C# using the double number format (because it is impossible to calculate the value of the objective function more accurately).

The results obtained are close to the results of a homogeneous search article [18].

7.3. Example 3

Here the space $X = [-4, 4]^{10}$, $x = (x_1, x_2, \dots, x_{10})$,

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

The f function is the well-known Rosenbrock test function used for local optimization methods. The function f takes the minimum value $f(x_*) = 0$ at the point $x_* = (1, 1, \dots, 1)$. The starting point of the search is $x = (-1.2, 1, -1.2, 1, \dots, 1)$ and $f(x) = 121$. The search for algorithm 2 with the parameters $\nu = 10^{-16}$, $\Gamma = 4$, $m = 100$ and $N = 10^7$ finds the minimum value of the objective function 3.1×10^{-29}. The search execution time was 3.3 seconds.

The results obtained are close to the results of a homogeneous search article [18]. But there the search execution time was 1.8 seconds. The normal probability distribution used in the search under study is somewhat more difficult to model than the uniform distributions in d-dimensional cubes used in [18].

7.4. Example 4

Let us consider an example with a very simple objective function, but in an optimization space of very large dimensionality for random search methods. Here, the space $X = \mathbb{R}^{1000}$, $x = (x_1, x_2, \dots, x_{1000})$, $f(x) = \sum_{n=1}^{1000} x_n^2$. The function f takes the minimum value of $f(x_*) = 0$ at the single point. The starting point of the search is $x = (1, 1, \dots, 1)$. The search for algorithm 2 with the parameters $\nu = 10^{-84}$, $\Gamma = 1$, $m = 100$ and $N = 10^6$ finds the minimum value of the objective function 3.7×10^{-163}. The search execution time was 30 seconds.

Homogeneous search of the article [18] with $N = 10^6$ received the minimum value of the objective function 1.6×10^{-14}. The search execution time was 13 seconds. In this example, the search for algorithm 2 turned out to be much more precise than the search for algorithm 1 of article [18].

8. Conclusion

The results show that the presented algorithm of inhomogeneous search, based on the use of the normal probability distribution, is quite effective. The presented random search program can be successfully used for solving optimization problems. The program itself is easy to use, and the choice of search parameters is not a difficult task. At the same time, the program allows you to solve 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 Zhiglyavsky A A 1983 On the random search of the global extremum *The Theory of Probability and Its Applications* **1** 129–136

[2] Ermakov S M, Zhiglyavsky A A and Kondratovich M V 1989 On Comparison of Certain Procedures for the Random Search for 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 under almost arbitrary noise* (Moscow: Science)

[9] Sushkov Yu A 1972 On one method of organizing a random search *Analysis of Operations and Statistical Modeling* Leningrad: Leningrad State University Publishing Vol **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 2010 On the convergence rate of the simulated annealing algorithm *Computational Mathematics and Mathematical Physics* Vol **50 1** 19–31
[16] Tikhomirov A S 2011 Lower bounds on the convergence rate of the Markov symmetric random search *Computational Mathematics and Mathematical Physics* Vol **51 9** 1524–1538
[17] Tikhomirov A S 2011 On the rate of convergence of one inhomogeneous Markov algorithm of search for extremum *Bulletin of St. Petersburg State University. Mathematics* Vol **44 4** 309–316
[18] 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
[19] Tikhomirov A S 2016 Markov Monotonous Search computer program *Certificate of State Registration of Computer Program no. 2016663645*