On the program implementation of a Markov homogeneous monotonous random search algorithm of an extremum

A S Tikhomirov

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 homogeneous monotonous random search algorithm of an extremum 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 $f: \mathbb{R}^d \rightarrow \mathbb{R}$ takes a minimum value at a single point $x_\star$. Consider the problem of finding the global minimum point $x_\star$ with specified accuracy $\varepsilon$. One way to solve this problem is to apply random search algorithms for the extremum of a function (see [1-17]). Such methods have long been used successfully in solving complex optimization problems. A theoretical study of the rate of convergence of certain Markov search algorithms was carried out in [3, 11-17]. This paper is a continuation of [12-14] and is devoted to a computer program [18] realizing one of the obtained algorithms for a homogeneous Markov monotonic search for an extremum.

2. Statement of the problem
As an optimization space, we consider the space $X = \mathbb{R}^d$ with metric

$$\rho_\infty(x, y) = \max_{1 \leq i \leq d} |x_i - y_i|,$$

where $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$. Closed ball of radius $r$ with the centre in point $x$ we denote by $B_r(x) = \{y \in \mathbb{R}^d : \rho_\infty(x, y) \leq r\}$. By $\mu$ denote $d$-dimensional Lebesgue measure. Metrics $\rho_\infty$ is chosen for reasons of simplicity of modeling of the considered random search. The fact is that the simulation of the search is based on modeling of uniform distributions in balls. A ball in the metric (1) is a cube. Simulating a uniform distribution in a multidimensional cube is simple, while it is quite difficult to effectively model a uniform distribution in an "ordinary" ball given by the Euclidean metric.

To search for the minimum point, we use a homogeneous Markov monotonic random search (see [3, 11-14]), described later with the aid of the modeling algorithm. Notation "$\eta \leftarrow P(\cdot)$" reads like this: “get the realization of a random vector $\eta$ with distribution $P$”. For numbers and points in the optimization space, operations of the form $k \leftarrow 1$ и $\xi \leftarrow x$ denote the usual assignment operations.

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

Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

Published under licence by IOP Publishing Ltd
Step 2. $\eta_k \leftarrow P(\xi_{k-1}, \cdot)$. 

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 $k < N$, then $k \leftarrow k + 1$ and go to step 2, otherwise finish the algorithm.

Here $x$ is a starting point of search, $N$ is a number of search steps, $P(x, \cdot)$ is Markov transition function (see [3, 11–14]).

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

On the second step of Algorithm 1 we choose a new “test” point $\eta_k$ in the optimization space. A new “test” point is chosen randomly, using the distribution $P(\xi_{k-1}, \cdot)$. Distribution $P(\xi_{k-1}, \cdot)$ depends on the position of the “old” search point $\xi_{k-1}$. This dependence makes it possible to improve the efficiency of random search. The transition function $P(x, \cdot)$ we will call the trial transition function.

We will consider the search, the trial transition function $P(x, \cdot)$ which does not depend explicitly on the step number $k$. Such a search is called homogeneous. A search whose trial transition functions clearly depend on the step number $k$, is called nonhomogeneous. Of course, due to heterogeneity, you can improve the search efficiency. But such a relationship complicates the choice of search parameters, and the “right” choice of search parameters can be a very difficult task (see, for example, [7]). Therefore, for the sake of ease of use of the search, we confine ourselves to a homogeneous search.

In the third step of Algorithm 1 we compare a new test point $\eta_k$ with old search point $\xi_{k-1}$. If a new test point $\eta_k$ no worse than the old search point $\xi_{k-1}$ (that is, if inequality $f(\eta_k) \leq f(\xi_{k-1})$ holds), then the search goes to a new point $\eta_k$ (the operator $\xi_k \leftarrow \eta_k$), otherwise the search remains at the old point (the operator $\xi_k \leftarrow \xi_{k-1}$).

In the fourth step of Algorithm 1, we check the search stopping condition. 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.

Note that the second, third, and fourth steps of Algorithm 1 are repeated $N$ times. The first step of Algorithm 1 is performed only once. We also note that the introduced random search is monotone, in the sense that inequalities $f(\xi_k) \leq f(\xi_{k-1})$ are performed for all $k \geq 1$.

3. Simulation of random search

The key question of choice for the type of search being investigated is the choice of the type of trial transition function $P(x, \cdot)$. When choosing a transition function, two criteria are usually used. First, the search must be sufficiently effective (requiring not too many steps to solve the task). In addition, the simulation of the distribution $P(x, \cdot)$ should be simple enough.

We use a homogeneous Markov monotonic random search, the transition function $P(x, \cdot)$ which minimizes the estimate of the complexity of random search investigated in [3, 12–14] when optimizing the simplest objective function. This search has several advantages. On the one hand, this search (for nondegenerate objective functions) ensures a good order of dependence of the obtained labor intensity estimates on $\varepsilon$ (see [3, 12–14]). On the other hand, this search is easily modeled. And the given examples of the use of the program show an acceptable efficiency in the optimization of not too complicated objective functions.

Consider a random search for Algorithm 1, trial transition functions $P(x, \cdot)$ which have symmetric densities of the form $p(x, y) = g(\rho_\infty(x, y))$, where $\rho_\infty$ is metric, and $g$ is nonincreasing nonnegative function defined on the semiaxis $(0, +\infty)$. Function $g$ we call a search form. Suppose that $0 < \nu < \gamma \leq \Gamma$ ($\nu$, $\gamma$, $\Gamma$ are search parameters) search form $g$ is given by the following formula:
where $\Lambda = 2^d (d \ln(\gamma / \nu) + \Gamma^d / \gamma^d)$ is normalizing constant (providing the necessary normalization condition for the density), $d$ is the dimension of the optimization space.

By $U_r(x, \cdot)$ we denote the uniform distribution in the ball $B_r(x)$, i.e. we put $U_r(x, \cdot) = \mu(\cdot \cap B_r(x))/\mu(B_r(x))$. Let $\alpha$ be uniformly distributed on a segment $[0, 1]$ random value. Let $p = d 2^d \ln(\gamma / \nu) / \Lambda$ and $q = \Lambda 2^{-d} / d$. Using the results of [12], we obtain the following algorithm for modeling a random vector $\eta$, having a distribution $P(x, \cdot)$ with the form (2).

Modeling Algorithm 1

Step 1. Get $\alpha$.

Step 2. If $\alpha \geq p$, then $r \leftarrow \Gamma$, otherwise $r \leftarrow \nu \exp(\alpha q)$.

Step 3. $\eta \leftarrow U_r(x, \cdot)$, complete the algorithm.

Since in the case of a metric $\rho_e$, simulate distribution $U_r(x, \cdot)$ very simple, then, in general the simulation algorithm 1 is very easily programmed.

By virtue of the recommendations of [12], we choose the value $\gamma = \Gamma / 2^{d/\nu}$. Thus, we use the distribution $P(x, \cdot)$ with the form (2) and two parameters $\nu$ and $\Gamma$ for modeling random search. In fact (see [12]), the distribution $P(x, \cdot)$ with form (2) is a mixture of uniform distributions in balls with center at the point $x$ and radii from the segment $[\nu, \Gamma]$. In the modeling algorithm 1, we first simulate the radius of the ball, and then simulate a uniform distribution in a ball with a selected radius. The presented simulation algorithm is only slightly more complicated than the algorithm for simulating the simplest random search (the so-called “blind search” [3, 5]) in which the uniform distribution in the pre-fixed area of the optimization space is used. The modeling algorithm 1 is simpler than most of the random search modeling algorithms used [3, 5-7].

4. Description of the program

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

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

For calculations in the program, a numeric double type is used, providing an accuracy of 15-16 characters. Note that this numerical format limits the possible accuracy of the solution of the problem. Arguing somewhat simplistically, we obtain the following conclusions. If the target function behaves approximately as a quadratic function in the vicinity of the global minimum, then with accuracy of approximation by argument in the order $10^{-8}$, we obtain the accuracy of approximation by the value of function in the order $10^{-16}$. If the minimum value of the objective function is different from zero, then the numeric double type, providing an accuracy of 15-16 characters, will not allow us to calculate the value of the function with accuracy higher $10^{-16}$. Thus, the typical accuracy of the solution of the
The problem will be of the order $10^{-7}$ of the approximation with respect to the argument, and of the order $10^{-14}$ of approximation by the value of the function. Such accuracy, as a rule, is sufficient from a practical point of view. And this accuracy of the solution of the problem can be obtained by using the considered random search program 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 at zero, then the problem can be solved with much higher accuracy.

To apply the search, you need to specify the target function, the search form parameters, the starting point of the search, and the number of search steps. The search results will be the end point of the search (approximating the point of the global minimum) and the value of the objective function at the end point of the search.

The submitted search has only four parameters. Parameters of the search form are positive numbers $\nu$ and $\Gamma$, for which inequalities must hold $0 < \nu < \Gamma$. The choice of search parameters depends on the success of the search and its laboriousness. Value $\nu$ can be chosen close to the required accuracy of the solution of the problem when approximating with respect to the argument. Value $\Gamma$ can be chosen close to or the expected accuracy of the initial approximation (the distance from the initial point of search to the minimum point) or to the diameter of the investigated region in the optimization space. When choosing $\Gamma$ one can use the upper bound.

The third parameter $N$ is the number of search steps. It is desirable to take the number of search steps large enough. When solving a single task, you can, for example, execute a billion search steps, even if the task is fairly simple and can be solved much faster. Modern personal computers may well perform similar amounts of computation, at least for not too complex target functions. True, for such amounts of calculation, the code of the target function must be programmed.

In addition, you must select the starting point of the search. It is clear that the starting point is better located closer to the point of global extremum.

Numbers $\nu$, $\Gamma$, $N$ and the starting point of the search is easy to set in the main program window.

It is more difficult to specify an objective function. The target function can be defined in two ways. First, you can write the function code directly in the program code (in C#). To do this, open the source project in Microsoft Visual Studio Professional 2010 (or a compatible version). In the project, you need to open the ObjectiveFunction.cs file. In this file, you need to specify the dimension of the optimization space and the code that calculates the value of the objective function. Then you need to recompile and run the program (for example, by pressing the F5 key). C# language is an object-oriented programming language, oriented to programming using classes. The ObjectiveFunction.cs file contains the definition of the ObjectiveFunction class, which contains only the definition of the objective function and the dimension of the optimization space. There is no other complex code in this file. When writing code that calculates the value of an objective function, as a rule, minimal information about a programming language such as C, C++, C#, Java is sufficient.

Second, the target function can be specified in the search program itself (without using Microsoft Visual Studio). To do this, click the button “Set the formula” in the program, select the “Use formula” option in the dialog box, specify the dimension of the optimization space, and write the formula that defines the objective function.

The method of specifying the objective function is shown in the text field with the inscriptions “We use the function code” or “We use the formula”.

The size of the optimization space is specified when specifying a function (or a constant in the ObjectiveFunction.cs file or a parameter when specifying a formula).

The rules for writing the formula are as follows. The formula can contain numbers, point coordinates, parentheses, arithmetic signs, function calls and constants. A decimal notation of numbers with a decimal separator is used (you can use both a comma and a period).

The coordinates of the point are written in the form $x[k]$ (or $X[k]$). The index $k$ must be a nonnegative integer constant, which is smaller than the dimension of the optimization space. For example, you can write $x[0] + x[1]$. 
Eight symbols of arithmetic operations are used: +, -, (unary and binary minus), *, /, % (% operator calculates the remainder after dividing the first operand by the second), ^ (exponentiation), binary operation | computes the maximum of two numbers, the binary operation & computes a minimum of two numbers.

The operations of calculating the maximum and minimum have the lowest priority, followed by addition and subtraction operations, then the operations of multiplication, division and calculation of the remainder, then there is an exponentiation, then a unary minus.

You can use 18 functions. C# names are used (for example, Math.Exp), abbreviated names (without the Math class), and the Russian variants of writing function names. For example, the natural logarithm can be written in the following three ways: Math.Log, log, ln. In names you can use both uppercase and lowercase letters (they are equivalent). Function arguments are written in parentheses. A complete list of functions is provided in the “Help” menu of the program.

In the formula, you can use two constants, these are the numbers π and e.

As usual, you can include spaces and newline characters in the formula. If you set the value of “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.

In the program, you can specify output formats for the value of the objective function and the point coordinates (the C# formatting rules are used). The format E specifies an exponential representation of numbers. The format F specifies the representation of numbers with a fixed comma (period). The format G specifies the shorter of the two E or F formats. Precision specifies the number of digits in the fractional part for E and F, and the number of significant digits for G.

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 either by a value depending on the system time of the computer, or by a given value. The program can store data in XML format, and export key search characteristics in text format. Note that using the old Microsoft Visual Studio 2010 development environment when writing the program allows you to work with the project even for users of computers running the operating system Windows XP.

5. 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]. In this case, 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. We cite a quote from [7] relating to the ultrafast annealing method proposed by L. Ingber: “Among the shortcomings of this method, one can call it that, in view of the large number of parameters, it sometimes takes several months to fine-tune it for a specific task.” At the same time, with proper selection of parameters, the method of ultrafast annealing can show very good results [6, 7].

The proposed search algorithm is largely free from insurmountable difficulties in the choice of parameters. To apply the search, you need to specify two parameters of the search form, the starting point of the search and the number of search steps. Considerations for choosing the search parameters described in the previous section can be considered quite simple. In particular, in the numerical examples of the next section, the minimum parameter selection was performed, consisting of just a few attempts to start the program with different parameter values.

6. Examples of using the program
Here are some examples of using the presented program to solve optimization problems. For computing, a personal computer with an Intel Core i5-4460 processor was used.

6.1. Example 1
We use an example from [5]. Here $X = \mathbb{R}^2$, $x = (x_1, x_2)$,
The objective function takes the form:

\[ f(x) = f(x_1, x_2) = x_1^4 + x_1^2 + x_1 x_2 + x_2^2. \]

Function \( f \) takes a minimum value at a single point \( x_* = (0, 0) \) and \( f(x_*) = 0 \). The starting point of the search is selected \( x = (1, 1) \) and \( f(x) = 4 \). The number of search steps (the number of calculations of the objective function) \( N \) here takes the value 10000.

Algorithm B of the book [5] receives 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] receives 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 it is a more complex search option, in which the movement performed in the previous step of the algorithm is taken into account when constructing a new search point.

Algorithm 1 with parameters \( \nu = 2 \times 10^{-24} \) and \( \Gamma = 0.7 \) receives the minimum value of the objective function \( 9.9 \times 10^{-69} \). In this example, the search for algorithm 1 turned out to be much more accurate than algorithms B and C of the book [5].

For the same objective function and search with parameters \( N = 10^6 \), \( \nu = 10^{-165} \), \( \Gamma = 0.7 \) we obtain the value of the objective function equal to zero (i.e., less than the value \( 5 \times 10^{-24} \), which defines a range of values of the double type of the C# programming language) and a minimum point \((7 \times 10^{-163}, -3 \times 10^{-163})\). The execution time of a million search steps was 0.06 seconds. Note that in this case, the extreme accuracy with which you can perform calculations on C# when using the numeric format double (due to the fact that you cannot more accurately calculate the value of the objective function) is achieved.

6.2. Example 2

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

\[ f(x) = f(x_1, x_2) = \frac{1}{2} \left( x_1^4 - 16x_1^2 + 5x_1 + x_2^4 - 16x_2^2 + 5x_2 \right). \]

Function \( f \) has four local minima, one of which is global. The starting point of the search is selected \( x = (4.0, 6.4) \) and \( f(x) = 537.18 \). Search for Algorithm 1 with parameters \( \nu = 10^{-8} \) and \( \Gamma = 10 \) when it finds the minimum value of the target function \(-78.323314075428\) and the minimum point \((-2.903534, -2.903534)\). Note that here we have reached the extreme accuracy with which we can perform calculations on C# using the numerical format double (because we can not calculate the value of the objective function more accurately).

6.3. Example 3

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

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

Function \( f \) is a known test function of Rosenbrock, used for local optimization methods. Function \( f \) assumes the minimum value \( f(x_*) = 0 \) in point \( x_* = (1.1, \ldots, 1) \). The starting point of the search is selected \( x = (-1.2,1, -1.2,1, \ldots, 1) \) and \( f(x) = 121 \). Search for Algorithm 1 with parameters \( \nu = 10^{-17} \) and \( \Gamma = 4 \) with \( N = 1000000 \) finds the minimum value of the objective function \( 2.84483 \times 10^{-29} \). The search time was 1.8 seconds.
6.4. Example 4
Consider an example with a very simple objective function, but in an optimization space of very large dimension for random search methods. Here the space \( X = \mathbb{R}^{1000} \), \( x = (x_1, x_2, \ldots, x_{1000}) \), \( f(x) = \sum_{n=1}^{1000} x_n^2 \).

Function \( f \) assumes the minimum value \( f(x_c) = 0 \) in a single point. The starting point of the search is selected \( x = (1, 1, \ldots, 1) \). Search for Algorithm 1 with parameters \( \nu = 10^{-10} \) and \( \Gamma = 10 \) at \( N = 1000000 \) finds the minimum value of the objective function \( 1.55934 \times 10^{-14} \). The search time was 13 seconds.

7. Conclusion
The obtained results show that 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. In this case, the program allows you to solve problems with the maximum accuracy, which can be obtained by using the numerical format double of the C# programming language.

References
[1] S.M. Ermakov, A.A. Zhiglyavsky, On the random search for a global extremum, Theory of Probability and its applications. No. 1 (1983) 129-136.
[2] S.M. Ermakov, A.A. Zhiglyavsky, M.V. Kondratovich, On the comparison of some procedures for the random search for a global extremum, Journal of Computational Mathematics and Mathematical Physics. Vol. 29, No. 2 (1989) 163-170.
[3] A. Zhigljavsky, A. Zilinskas, Stochastic global optimization, Springer-Verlag, Berlin, 2008.
[4] A. Zhigljavsky, A. Zilinskas, Stochastic global optimization: a review on the occasion of 25 years of Informatica, Informatica, Vol. 27, No. 2 (2016) 229-256.
[5] J.C. Spall, Introduction to stochastic search and optimization: estimation, simulation, and control, Wiley, New Jersey, 2003.
[6] L. Ingber, Very fast simulated re-annealing, Mathl. Comput. Modelling. V. 12 (1989) 967-973.
[7] A.S. Lopatin, Simulated Annealing Method, Stochastic optimization in computer science. Issue 1 (2005) 133-149.
[8] O.N. Granichin, B.T. Polyak, Randomized estimation and optimization algorithms under almost arbitrary noise, Science, Moscow, 2003.
[9] Yu.A. Sushkov, On a method for the organization of random search, Operations research and statistical modeling. Leningrad University Publishing House, Leningrad, Vol. 1 (1972) 180-186.
[10] D. Tarlowski, On the convergence rate issues of general Markov search for global minimum, Journal of Global Optimization. Vol. 69, Issue 4 (2017) 869-888.
[11] V.V. Nekrutkin, A.S. Tikhomirov, Speed of convergence as a function of given accuracy for random search methods, Acta Applicandae Mathematicae. 33 (1993) 89-108.
[12] A.S. Tikhomirov, On the Markov homogeneous optimization method, Computational Mathematics and Mathematical Physics, Vol. 46, No. 3 (2006) 361-375.
[13] A.S. Tikhomirov, On the convergence rate of the Markov homogeneous monotone optimization method, Computational Mathematics and Mathematical Physics. Vol. 47, No. 5 (2007) 780-790.
[14] A. Tikhomirov, T. Stojunina, V. Nekrutkin, Monotonic random search on a torus: integral upper bounds for the complexity, Journal of Statistical Planning and Inference. Vol 137, Issue 12 (2007) 4031-4047.
[15] A.S. Tikhomirov, On the convergence rate of the simulated annealing algorithm, Computational Mathematics and Mathematical Physics. Vol. 50, No. 1 (2010) 19-31.
[16] A.S. Tikhomirov, Lower bounds on the convergence rate of the Markov symmetric random search, Computational Mathematics and Mathematical Physics. Vol. 51, No. 9 (2011) 1524-1538.
[17] A.S. Tikhomirov, On the rate of convergence of one inhomogeneous Markov algorithm of search for extremum, *Vestnik St. Petersburg University. Mathematics*. Vol. 44, No. 4 (2011) 309-316.

[18] A.S. Tikhomirov, Computer program “MarkovMonotonousSearch”, Certificate of state registration of the computer program No. 2016663645 (2016).