# Method of UNIT testing for algorithms of computing software modules

Kovartsev A.N., Popova-Kovartseva D.A., Gorshkova E.E.

Samara State Aerospace University

**Abstract.** The method of automating Unit testing processes for computing software modules is considered in the paper. Modern means of testing automation, which are analyzed in many scientific studies, specialize mainly in testing graphical user interfaces, web-interfaces, network communications, information systems, etc., which is the result of a huge demand in the market for software products within these spheres. Software modules of computing character are overshadowed by such products despite the fact that these modules have considerable scientific and practical value. They deal mostly with high tech spheres: aerospace cluster, energy industry, defense complex, etc. The article presents an original method of Unit testing for computing modules based on the algorithm of global search for infinite discontinuity points of testing function, which allows to find fatal errors in computing software modules, as well as incorrectness in implementation of algorithms for mathematical models. The universal method of Unit testing is offered within the class of computing modules, which helps to minimize the time for program debugging and to find fatal errors with less effort, as well as to organize total module testing for all modules of the program.

**Keywords:** Unit testing, testing automation, computing software modules, global optimization, fatal errors

**Citation:** Kovartsev A.N., Popova-Kovartseva D.A., Gorshkova E.E. Method of unit testing for algorithms of computing software modules. Proceedings of Information Technology and Nanotechnology (ITNT-2015), CEUR Workshop Proceedings, 2015; 1490: 252-261. DOI: 10.18287/1613-0073-2015-1490-252-261

## Introduction

Developers of modern software (SW) face the challenge to carry out their projects within tight deadlines and with minimal resources consumption [1, 2], whereas software vendors strive to carry out testing appropriately, quickly, and thoroughly. Most types of work, used in software programming, are to be supported by automated means of testing.

Modern means of automated testing are constantly expanding at present and include testing of graphical user interfaces, checking requirement compatibility, download speed, code coverage, web-interface, network communications, information

systems, etc. This results from the huge demand in the market for software products within these spheres. Software modules of computing character are overshadowed by such products despite the fact that these modules have considerable scientific and practical value. They deal mostly with high tech spheres [3]: aerospace cluster, energy industry, defense complex, etc.

Testing is the method of providing the required level of SW quality. In the general sense, software testing is the process that allows to determine the correctness, completeness and quality of the developed software product. Unfortunately, it is impossible to determine unambiguously whether the analyzed program functions correctly or not, as well as to guarantee the absence of defects in a software product because human factor problems may appear at all SW lifecycle stages [4]. Therefore, all existing testing methods operate within private formal methods of testing organized for the analyzed product. The list of modern methods and approaches for solving the problem of program testing is extensive and diverse. On the one hand, this diversity is determined by the current practice of using a computer while solving a variety of problems and, accordingly, by the specific features of software products themselves.

The class of computing software modules (CSM), based on the use of mathematical models, has some specific features. Such programs, carried out on a computer, are sure to calculate any function that realizes the display output of input data. This implies that a computer by means of its resources finishes the definition of a partly determined function, which results in complete definition. Consequently, it is possible to estimate if the results of program execution are right or wrong only by comparing the specification of the expected function with the results of its calculation; this is carried out in the testing process. For the class of computing software modules, methods of Unit testing [20] and test tools based on models using formal methods [4,6] are more suitable.

Formal methods usually allow to solve a limited range of testing software tasks within a particular class, however, they are able to work effectively in industrial projects and require a minimum number of special skills and knowledge so that to be used [4]. Currently, within monitoring the formal properties of SW, methods of test construction are thoroughly developed on the basis of finite-state automation [7, 8]. For them, accuracy characteristics and evaluation of completeness of performed tests are known. In these techniques, computing route is analyzed not only for performing some formulae, but also for coordination with the specified automation model of the proper behavior. Nowadays these methods are rarely used, mainly for monitoring small critical applications [4]. Formal methods are rather “heavy-weight”, they require well-qualified specialists, at least at the stage of software modeling. The weak point of formal methods is the need to construct the model itself on the parallel basis and to check its correctness.

The practice of industrial software production shows that the most effective strategy is to search and correct as many errors as possible at the earliest stage of software development. Methods of Unit testing are appropriate for this purpose. Unit testing consists in checking the software performance on a set of input and corresponding output data. This approach allows to reveal a significant number of errors and locate them quickly as it is easier to find an error within a module than to do it within the whole project. CSM can be easily interpreted with a vector

computable function $Z = f(X) = (f_1(X), f_2(X), \dots, (X))^T$, having a set of input parameters $X = (x_1, x_2, \dots, x_n)^T$ of the module and a set of calculated data $Z = (z_1, z_2, \dots, z_m)^T$.

The origin of errors for CSM is extensive and diverse. Errors occur when initial and boundary conditions, which characterize the value and location of external factors, are set incorrectly. They may be related with a set of limitations and assumptions resulting from the physical nature of the object and limiting the range of input parameters.

Among the errors of this kind for CSM, two most common groups can be distinguished: calculation errors and logical (algorithmic) errors. Calculation errors are mostly connected with incorrect recording or programming of mathematical expressions and manifest themselves as arithmetic error of division by 0, the square root of a negative number, as calculation of rational or transcendental functions, etc. They can only be detected while executing the program and lead to program stoppage.

Logical errors are connected with the distortion of problem solving algorithm and result from incorrect problem setting, wrong consideration of all conditions for solving the problem, incorrect management organization within CSM, and errors in the input of logical expressions. These errors are difficult to correct, corrections often being made with the help of formal methods [5].

Nevertheless, there is a universal method for detecting errors of calculating and partly logical character in computing software modules [3, 9]. This method is based on the fact that all computing modules, implemented as a program, "work" within integrity of used functions or mathematical models. Otherwise, the program can't be used. If we organize the search for infinite discontinuity of the function by some means, we will be able to detect all calculating errors of any origin mentioned above. Relatively "simple" methods of global optimization (GO) of multivariable functions are appropriate for this purpose[10].

In this paper we consider the improved algorithm of global search for points of discontinuity of the second type, intended for detecting error situations in the software modules of computing character. This algorithm is the basis of Unit testing for computing software modules.

## 1. Problem Statement

Let the area of error search in computing module be a unit cube (in general – hypercube) $\Pi = [0, 1] \times [0, 1] \times [0, 1]$, which is proportionally divided into eight smaller cubes.

Unit testing algorithm of computing modules, formally described by vector function $Z = f(X)$, can be put as the problem of global optimization

$$\max_X f_k^2(X), \quad k = 1,\dots,m, \tag{1}$$

which is aimed at detecting (or ensuring the absence of) discontinuity points of the second type in the examined function. The infinite discontinuity $\pm\infty$ is realized in numbers marked by code NaN on a computer; in fact we can specify the upper limit

of acceptable values $|f_k(X)| \le M_{sup}^{(k)}$ or each calculating parameter of a module. To simplify the situation, we will further consider scalar function $f(X) = max f_i^2(X)$.

Among the well-known one-parameter methods of multiextremal optimization, R.G. Strongin statistic information method is the most effective [10]. The method is based on the use of approximate posterior probability distribution of global extremum location, which is formed in the process of function testing, which allows to realize a more balanced strategy to search for function global minimum. This strategy is so effective that it is often transferred from one-dimensional case to multivariable function optimization.

It is shown in the paper [10] that function extremum search is realized by maximizing a simple characteristic function:

$$R(i) = \mu(x_i - x_{i-1}) + \frac{(z_i - z_{i-1})^2}{\mu(x_i - x_{i-1})} - 2(z_i + z_{i-1}), \quad (2)$$

where $\mu$ – estimation of Lipschitz constant, which is calculated in the process of function extremum search:

$$\mu = \begin{cases} 1 & M = 0, \\ rM & M > 0, \end{cases} \qquad M = \max_i \frac{|z_i - z_{i-1}|}{(x_i - x_{i-1})},$$

where $r$ – parameter.

The condition of Lipschitz for optimized function simplifies greatly the search for function extremum as the limitation of function growth degree allows to find local extremum vicinity quickly. However, while searching for a set of infinite discontinuity, areas of function monotony as well as extremum vicinity are equally useless. It is much more important to determine the criteria which will be responsive to the fast increase and decrease of function.

Nowadays, mathematical aspects of function behavior in vicinities of discontinuity points are not analyzed thoroughly, which complicates detecting function discontinuity presence while analyzing its behavior on local continuous sections. The paper [11] offers the characteristic function, created by analogy with (2), but it is more adapted to solve the task of search for infinite discontinuities points. The following characteristic function is offered to use:

$$R(i) = (d^2 f(X_i^c))^2 D_i^r, \quad (3)$$

where $X_i^c$ – the centre of a cube, $D_j$ – cube diagonal of search algorithm, $r$ – scaling parameter.

The second differential of function can be calculated with the help of interpolation of initial function using Newton's first interpolation formula [12] for full factorial plan $3^n$ [13]. Then, using $P_{2,2,2}(X)$ it is easy to calculate $d^2 f(X) \approx d^2 P_{2,2,2}(X)$.

In search algorithm, there is a proportional division of search area into smaller parts: the unit interval – into 2 parts; square – into 4 parts, cube – into 8 parts, etc. Fig. 1 shows a diagram of search area division for two-dimensional case. We will consider the three-dimensional case further in order to present a good illustration.

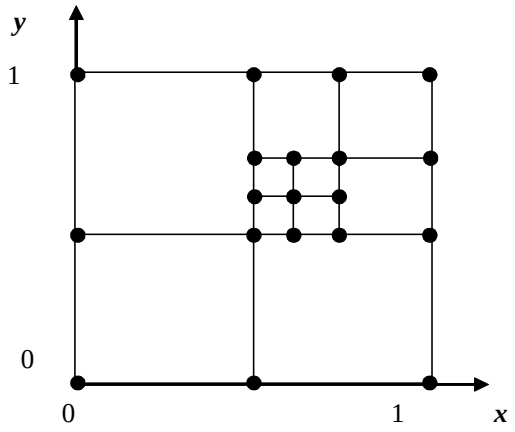



**Fig. 1.** – Diagram of Search Area Division

Construction of interpolating polynomial for multidimensional case can be realized as follows.

$Z = \|z_{ijk}\|(i,j,k = 0,1,2)$

is the matrix of testing function values at nodal points of full factorial plan $3^n$. We introduce new variables $q = \frac{x_1 - x_1^{(0)}}{h}; p = \frac{x_2 - x_2^{(0)}}{h}; r = \frac{x_3 - x_3^{(0)}}{h}$. The starting point of interpolation grid is $X^{(0)} = (x_1^{(0)}, x_2^{(0)}, x_3^{(0)}$; $h$ – the step of interpolation grid. Newton's interpolation polynomial is to be calculated in the form of matrix. Symbols of multidimensional matrix and content of main operations are borrowed from the paper [13]. To do this, we need to calculate the matrix of finite differences $\tilde{Z}$.

$$\tilde{Z} = \left\| \begin{array}{ccc|ccc|ccc} z_\alpha & \Delta_r^1 z_\alpha & \Delta_{rr}^2 z_\alpha & \Delta_q^1 z_\alpha & \Delta_{qr}^{1+1} z_\alpha & \Delta_{qrr}^{1+2} z_\alpha & \Delta_{qq}^2 z_\alpha & \Delta_{qqr}^{2+1} z_\alpha & \Delta_{qqrr}^{2+2} z_\alpha \\ \Delta_p^1 z_\alpha & \Delta_{pr}^{1+1} z_\alpha & \Delta_{prr}^{1+2} z_\alpha & \Delta_{qp}^{1+1} z_\alpha & \Delta_{qpr}^{1+1+1} z_\alpha & \Delta_{qprr}^{1+1+2} z_\alpha & \Delta_{qqp}^{2+1} z_\alpha & \Delta_{qqpr}^{2+1+1} z_\alpha & \Delta_{qqprr}^{2+1+2} z_\alpha \\ \Delta_{pp}^2 z_\alpha & \Delta_{ppr}^{2+1} z_\alpha & \Delta_{pprr}^{2+2} z_\alpha & \Delta_{qpp}^{1+2} z_\alpha & \Delta_{qppr}^{1+2+1} z_\alpha & \Delta_{qpprr}^{1+2+2} z_\alpha & \Delta_{qqpp}^{2+2} z_\alpha & \Delta_{qqppr}^{2+2+1} z_\alpha & \Delta_{qqpprr}^{2+2+2} z_\alpha \end{array} \right\| \quad (4)$$

Here α= 000. We introduce a vector

$$\begin{aligned} Q &= (1 \quad q \quad q(q-1)/2)'; \\ P &= (1 \quad p \quad p(p-1)/2)'; \\ R &= (1 \quad r \quad r(r-1)/2)'. \end{aligned} \quad (5)$$

Considering $(\lambda, \mu)$-convoluted product of vectors *Q, P, R* with λ = 0, μ= 0, we get the matrix of independent variables

$\tilde{X} = QPR = \|q_i \cdot p_j \cdot r_k\| \quad (i,j,k = 0,1,2$ (6)

Thus, Newton's interpolation polynomial has the following matrix form

$$P_{2,2,2}(X) = {}^3\left(\tilde{Z}\ \tilde{X}\right) = \sum_{c_0,c_1,c_3=0}^{2} \tilde{Z}_{c_0c_1c_2} \cdot \tilde{X}_{c_0c_1c_2}, \quad (6)$$

where $c = (c_0, c_1, c_2)$ – Caylean summation index [14].

The elements of matrix (4) can be calculated using the definition of finite differences of corresponding orders.

To calculate partial derivatives we need vectors
$Q' = (0 \quad 1 \quad q - 0.5)'$; $P' = (0 \quad 1 \quad p - 0.5)'$; $R' = (0 \quad 1 \quad r - 0.5)'$; $Q'' = P'' = R'' = (0 \quad 0 \quad 1)'$,
then

$$P'_{x_1}(X) =^3 (\tilde{Z}(Q'PR))\frac{1}{h}; P'_{x_2}(X) =^3 (\tilde{Z}(QP'R))\frac{1}{h}; P'_{x_3}(X) =^3 (\tilde{Z}(QPR'))\frac{1}{h},$$

and the second derivatives:

$$P''_{x_1x_1}(X) =^3 (\tilde{Z}(Q''PR))\frac{1}{h^2}; P''_{x_2x_2}(X) =^3 (\tilde{Z}(QP''R))\frac{1}{h^2}; P''_{x_3x_3}(X) =^3 (\tilde{Z}(QPR''))\frac{1}{h^2}.$$

Now it is easy to calculate the value of the second differential in the centers of each eight cubes of original search area partition:

$$d^2 f(X_i^c) \approx (P''_{x_1x_1}(X_i^c) + P''_{x_2x_2}(X_i^c) + P''_{x_3x_3}(X_i^c) + 2(P''_{x_1x_2}(X_i^c) + P''_{x_1x_3}(X_i^c) + P''_{x_2x_3}(X_i^c)))h^2.$$

The value of characteristic function is calculated for each of new cubes, which are written onto the line ordered list in descending order of values. At each stage of search algorithm for discontinuity points of testing function, the first item on the list is chosen – a cube with maximum value of characteristic function, which is subjected to further division. The algorithm works till the condition of algorithm stoppage appears: $(\max|z_i| > M_{sup}) \vee (min_i h_i < \varepsilon)$. This condition of algorithm stoppage ensures the completion of its work if the value of testing function is outside function domain or the given density of viewing the original area of testing function is reached.

## 2. Examples of using the proposed method of Unit testing for computing modules

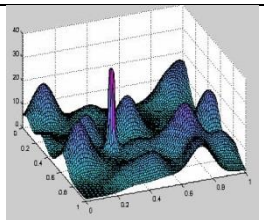
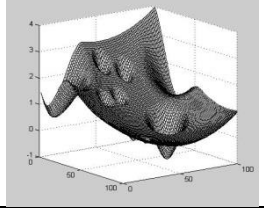
### 2.1. Method testing with model examples

The proposed testing method of CSM got its name as finite differences method (FDM) due to the use of finite differences of the function. The efficiency of its work can be evaluated by means of discontinuous function – Kovartsev's test [11], specially developed for this case, and a set of test functions generated by the GKLS generator [15]. The first test is characterized by a single discontinuity point (which is difficult to detect) added to linear combination of error functions. The second one is a continuous function with a large number of local extrema. Test functions are presented in Table 1.

In literature, the efficiency of search algorithms is usually evaluated using the operating characteristics machine [16]. Operating characteristic is the dependence of error detection probability $P_{alg}$ on the amount of calls to the tested function $N_f$.

Since the second-order discontinuity points can be found by any of the global optimization algorithms, the efficiency of the proposed FDM algorithm was compared with the efficiency of direct GO method, for example, the modified bisection method (BM) [17].

**Table 1.** A set of test functions

| № | Function | General view |
|---|---|---|
| 1 | Kovartsev test function:<br>$f(x_1, x_2) = \sum_{i=0}^{19} (i+1) e^{-\frac{(x_1 - a_{1i})^2 + (x_2 - a_{2i})^2}{0.01}} + 1/(1 - e^{-\frac{(x_1 - b_1)^2 + (x_2 - b_2)^2}{0.01}})$<br>$x_1, x_2 \in [0; 1]$<br>20 local extrema. One second-order discontinuity point |  |
| 2 | GKLS test functions.<br>Continuous twice differentiable function. 10 local extrema. One global extremum. Points of discontinuity are not observed. |  |

The operating characteristics of FDM and BM methods are shown in Fig. 2. They are created for test function 1 which has a local discontinuity point of the second type (see Table 1). As we can see from fig. 2, the efficiency of the proposed algorithm is much greater than the efficiency of the bisection method. In Fig. 2 the solid line indicates operating characteristics of FDM algorithm, the dashed one indicates characteristics of BM algorithm. It happens because the bisection method is focused on the optimization of continuous functions, which leads to a more detailed analysis of the function areas when Lipschitz constant evaluation increases. This situation occurs every time when the function is calculated near the points of its discontinuity. By contrast, FDM method is oriented on looking for areas of rapid growth of the test function.

The situation changes if FDM method "works" with continuous function. Figure 3 shows the operating characteristics of these methods, created for continuous GKLS test function.

The figure illustrates the fact that the efficiency of FDM algorithm for continuous functions is much lower than the efficiency of BM algorithm. If we have continuous functions with no discontinuity points of the second type (test software module has no errors), the finite difference method is forced to examine thoroughly the space of the optimized variables.

### 2.2. Testing of software modules for calculating acoustic characteristics of gas pressure regulator

This part presents the results of Unit testing for computing models included in the program that realizes the optimization of gas pressure regulator (GPR) parameters with use of orifice plates [21].Significant changes in pressure during orificing and speed acceleration generate the noise which accompanies the work of these machines. This noise exceeds the established health standards. Rational choice of orifice flow area (and their quantity) can significantly reduce the noise level of this device [18].

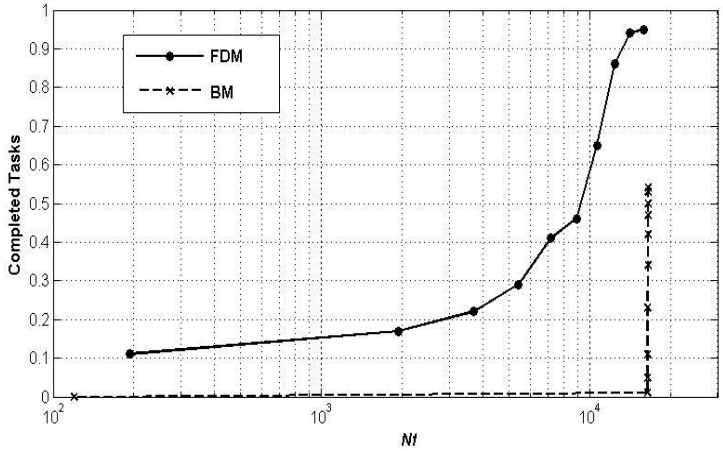



**Fig. 2.** – Operating characteristics of FDM and BM methods for function №1

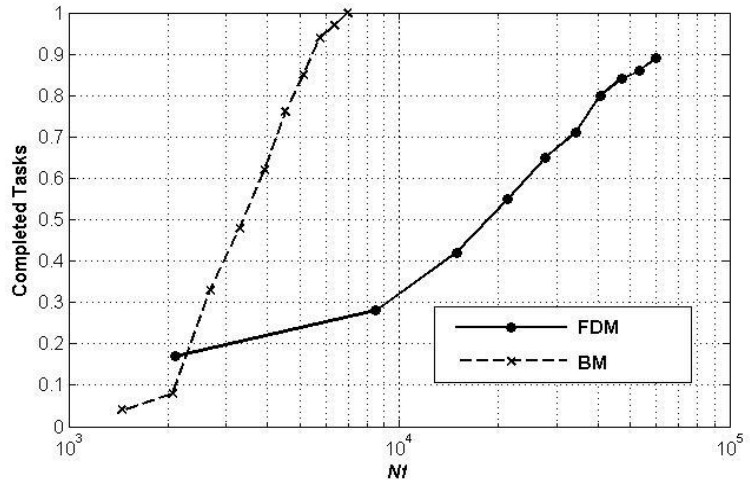



**Fig. 3.** – Operating characteristics of FDM and BM methods for function №2

Output parameters of GPR for the stationary case have been calculated by solving the system of nonlinear equations that describe gas motion in its specific sections: the valve mechanism and orifice package:

$$\begin{cases} G_x - G_1 = 0, \\ G_1 - G_2 = 0, \\ .... \\ G_{n-1} - G_{\mathrm{n}} = 0, \end{cases} \tag{7}$$

$G_x$– gas flow through the valve, $G_i$ – gas flow through $i$-orifice plate. When the input parameters of module are given, for example, $p_i, p_{i+1}$– pressure before and after orifice plate correspondingly; $S_i$ – areas of $i$-orifice flow, etc. we can calculate gas mass flux through orifice and as a result – acoustic power generated by orifice plate [19].

We used a rather simple module for calculating the orifice flow capacity ($C_{vi}$) in order to calculate the gas mass flux through orifice. Testing the module by FDM method with approximately 200 calls to the module revealed methodological error in the algorithm model. It turned out that if $p_i \to p_{i+1} - C_{vi} \to \infty$, and $p_i < p_{i+1}$, flow capacity is indefinite (error code NaN occurs).

Certainly, it would be possible to require the calculation with this module $C_{vi}(p_i, p_{i+1})$ to be carried out only in accordance with the condition $p_i > p_{i+1}$, which is natural for this type of device. But how can it be realized when numerical method for solving systems of nonlinear equations (8) generates the values of independent variables at each iteration on its own, not taking into consideration the above mentioned circumstances? The easiest way to solve the problem is the artificial replacement of function infinite discontinuity $C_{vi}(p_i, p_{i+1})$ with a larger but finite discontinuity, and extension of definition by "penalty" value where it is indefinite. In this case, the algorithm for solving systems of nonlinear equations is to find solutions on its own, "starting with" "dysfunctional" combinations of independent variables.

High speed of detecting fallible combinations of FDM independent variables in this example can be explained by the fact that at the first stage of work, when the area of rapid function growth is not detected, FDM distributes test points of testing function in the search area. Since function $C_{vi}(p_i, p_{i+1})$ has significant areas of uncertainty, FDM finds them quickly.

## Conclusion

The paper offers an original method of Unit testing for computing modules, based on the algorithm of global search for infinite discontinuity of the testing function, which allows to detect fatal errors in software computing modules, as well as incorrectness in implementation of mathematical models of algorithms.

The proposed scheme of accelerating algorithms for global optimization applied to the search for points of discontinuity of the second order has confirmed completely its viability with model and real examples. The basic idea of FDM algorithm is to introduce a new heuristic characteristic function to the classical algorithm of global optimization. The new function is based on the analysis of Strongin characteristic function and takes into account the problem matters being solved. FDM algorithm is the universal method of Unit testing for the class of computing modules. The application of this method leads to the reduction of time for debugging, helps to find fatal errors with less effort, and, as a result, to organize total testing of all program modules.

## References

1. **Dastin E, Reshka D, Paul D.** Automatic software testing. Application, operation and maintenance. Moscow: Lori, 2003; 567 p. [in Russian]
2. **Lipaev VV.** Program testing. M.: Radio and connection, 1986; 296 p. [in Russian]

3. **Kovartsev AN.** Automation of software development and testing. Samara State Aerospace University, 1999; 148 p. [in Russian]
4. **Kuliamin VV.** Software verification methods. Source: <http://panda.ispras.ru/~kuliamin/docs/VerMethods-2008-ru.pdf >
5. **Kovartsev AN.** An efficient algorithm for testing the truth of assertions for real numbers expressed in relational signatures. Computer Optics, 2014; 38(3): 550-554. [in Russian]
6. **Heitmeyer C, Archer M, Bharadwaj R, Jeffords R.** Tools for constructing requirements specifications: The SCR toolset at the age of ten. Journal of Computer Systems Science and Engineering, 2005; 20(1): 19-35.
7. **Farchi E, Hartman F, Pinter SS.** Using a model-based test generator to test for standard conformance. IBM Systems Journal, 2002; 41(1): 89-110.
8. **Cavalli A, Gervy C, Prokopenko S.** New approaches for passive testing using an Extended Finite State Machine specification. Information and Software Technology, 2003; 45(12): 837-852.
9. **Kovartsev AN, Logvinov AL.** Efficiency improvement of testing algorithms for computing modules. Journal of Telecommunication systems, 2004; 4. [in Russian]
10. **Strongin RG.** Search for global optimum. Moscow: Znanie, 1990. [in Russian]
11. **Kovartsev AN, Popova-Kovartseva DA, Serpovskaya EE.** Testing mathematical models of computing algorithms based on global optimization method. Information technology and nanotechnologies. SSAU, 2015; 191-196. [in Russian]
12. **Kovartsev AN.** Computational mathematical. Samara: Ofort, 2011; 230 p. [in Russian]
13. **Ermakov SM, Zhigliavsky AA.** Mathematical theory of optimal experiment operation. Moscow: Nauka, 1987; 320 p. [in Russian]
14. **Sokolov NP.** Introduction to multidimensional matrix theory. Kiev: Nukova dumka, 1972; 177 p. [in Russian]
15. **Gaviano M, Kvasov DE, Lera D, Sergeyev YaD.** Software for generation of classes of test functions with known local and global minima for global optimization. ACM TOMS, 2003; 29(4): 469-480.
16. **Gergel VP, Strongin RG.** Absolute. Software system for global optimization method studying. Textbook. Nizhny Novgorod: Nizhegorodsky University Press, 1998; 141 p. [in Russian]
17. **Kovartsev AN, Popova-Kovartseva DA, Abolmasov PV.** Efficiency study of global parallel optimization for multivariable function. Vestnik NNGU, 2013; 3(1): 252-261. [in Russian]
18. **Igolkin AA, Kruchkova AN, Koh AI, Safin AI, Shakhmatov EV.** Pressure reducing valve noise reduction. Proceedings of the Nineteen International Congress on Sound and Vibration (ICSV 19). The international institute of Acoustics and Vibration, 2012, July 08-12.
19. **Istvan L, Beranek V, Beranek L.** Noise and vibration control engineering. Second edition. Principles and applications. Published by Wiley and Sons, Inc, 2006; 966 p.
20. **Meszaros, G.** xUnit test patterns: rafactoring test code. U.S. at Courier in Wesrford, Massachusetts, 2009; 835 p.
21. **Karczub DG, Catron FW, Allen C.** Blow-down valve noise and interactions with down stream orifice plates. Fagerlund Proceedings of IMECE'03. ASME International Mechanical Engineering Congress, 2003; 43-49.