APPROXIMATION OF A MULTIDIMENSIONAL DEPENDENCY BASED ON A LINEAR EXPANSION IN A DICTIONARY OF PARAMETRIC FUNCTIONS[*]

M.G. Belyaev[1], E.V. Burnaev[2]

Abstract. We consider the problem of a multidimensional function approximation using a finite set of pairs "point"—"function value at this point". As a model for the function we use expansion in a dictionary containing non-linear parametric functions. Several sub-problems should be solve when constructing an approximation based on such model: extraction of a validation sample, initialization of parameters of the functions from the dictionary, tuning of these parameters. We propose efficient methods for solving these sub-problems. Efficiency of the proposed approach is demonstrated on some problems of engineering design.

Keywords: nonlinear approximation, parametric dictionaries.

1 Introduction

For engineering design we need to model complex physical phenomena. Typically used models are represented by complex systems of differential equations. Such systems do not have analytical solutions, so computationally heavy numerical methods are used. One approach to solving problems of engineering design, actively developing in recent years, is the surrogate modelling [1, 2]. In this approach a complex physical phenomenon is described by a simplified (surrogate) model constructed using data mining techniques and a set of examples, representing results of a detailed physical modelling and/or real experiments. The problem of approximation of a multidimensional function using a finite set of pairs "point"—"value of the function at this point" is one of the main problems to be solved in the construction of the surrogate model. We will consider this problem in the following formulation:

Problem 1 *Let $f(\vec{x}) \in R^1$ be some continuous function on a compact set $\mathrm{D} \subset R^d$ with known output values in a finite set of input points. The set $S_{learning} = \{\vec{x}_i, y_i\}_{i=1}^{N_{learning}}$, $\vec{x}_i \in \mathrm{D}$, $y_i = f(\vec{x}_i)$ forms the learning sample. The approximation problem is to construct an approximation $\hat{f}(\vec{x})$ (approximator) of the function $f(\vec{x})$ using the given data sample $S_{learning}$ such that $f(\vec{x}) \approx \hat{f}(\vec{x})$ for all $\vec{x} \in \mathrm{D}$.*

[*]The authors are partially supported by Laboratory for Structural Methods of Data Analysis in Predictive Modeling, MIPT, RF government grant, ag. 11.G34.31.0073; RFBR grant 13-01-00521. Results, described in this work, were obtained in the framework of "COPTI-X: Surrogate Model Construction for Structure Approximation and Optimization" joint project with Airbus Operations SAS.

[1]Institute for Information Transmission Problems RAS, Moscow Institute of Physics and Technology, Datadvance LLC, belyaev@iitp.ru

[2]Institute for Information Transmission Problems RAS, Moscow Institute of Physics and Technology, Datadvance LLC, burnaev@iitp.ru

Remark 1 *In general case values of the function $f\left(\vec{x}\right)$ are known only for the finite set of input points, so the proximity $f\left(\vec{x}\right) \approx \hat{f}\left(\vec{x}\right)$ is usually measured be the mean square error $Q\left(S_{test}, \hat{f}\right) = \frac{1}{N_{test}}\sum_{i=1}^{N_{test}}\left(y_i - \hat{f}\left(\vec{x}_i\right)\right)^2$ calculated using an independent test data sample $S_{test} = \{\vec{x}_i, y_i\}_{i=1}^{N_{test}}$, $\vec{x}_i \in \mathrm{D}$, $y_i = f\left(\vec{x}_i\right)$. The criterion $Q\left(S_{test}, \hat{f}\right)$ of approximation quality makes sense if input vectors from the samples $S_{learning}$ and S_{test} are generated by the same distribution and cover the design space D sufficiently densely. We call the function $Q\left(S, \hat{f}\right)$ for some set S of pairs "point"—"value of the function at this point" as error function on the set S.*

Due to requirements of surrogate modelling problems (in particular, the need to build quickly computable global approximation model and to work with large data samples) the most common method of solving approximation problems is based on Artificial Neural Networks (ANN) [3]. An approximation based on the ANN model provides high-speed calculations of output predictions. The ANN model can be easily "extended" by increasing number of layers and/or their sizes as the learning sample size increases while the computational complexity of the approximation model construction grows only linearly.

A typical scheme of approximation construction based on the ANN model can be divided into two phases: an initialization phase (setting the initial values of the model parameters and extraction of the validation sample) and a training phase (tuning the ANN parameters to fit the data sample). Usually random methods are used during the initialization phase. Such methods lead to a large scatter of the approximation accuracy. Surrogate models are constructed to replace computationally heavy objective functions (and/or constraints) in optimization problems, see the paper [4]. Engineering design based on surrogate models is iterative: after finding the optimum of the objective function (surrogate model) in the neighbourhood of the optimum additional pairs "point"—"value of the function at the point" are generated, the surrogate model is re-constructed, and then the model is optimized, etc. Therefore unpredictable significant changes in the surrogate model structure and significant variations of the approximation accuracy deteriorate surrogate based optimization.

This paper investigates methods for constructing approximations based on the linear expansion in nonlinear functions from the parametric dictionary (ANN model with one hidden layer). We propose methods for initialization and training that reduce the average approximation error and its variations. Let us describe the structure of the paper.

The model based on a linear expansion in a dictionary of parametric functions is described in Section 2.1, main steps of the approximation construction based on this model are described in Section 2.2. The next few sections are devoted to various sub-problems that arise when constructing an approximation using the proposed algorithm. In section 3 we consider sub-problems of the initialization step of the approximation construction algorithm. In sub-section 3.1 we describe a new algorithm for the validation sample extraction such that points from the validation sample are distributed as uniformly as possible among the remaining points of the learning sample. We propose a computationally efficient deterministic algorithm for the validation sample extraction based on the greedy optimization of some uniformity criterion. In sub-section 3.2 we describe a new algorithm for the initialization of functions from the parametric dictionary. In section 4 we consider a new training algorithm. In sub-section 4.1 we propose a special form of the error function, which takes into account the structure of the approximation error dependence on different groups of parameters and includes adaptive regularization. In sub-section 4.2 we describe a new method for the regularization parameter selection. Each of sections 3 and 4 contains results of the computational experiments on artificial functions. Experimental results for some engineering design problems are described in section 5.

2 Algorithm for approximation construction

2.1 Model based on a linear expansion in a dictionary of parametric functions

We model the approximation $\hat{f}(\vec{x})$ by the linear expansion in a dictionary of parametric functions, i.e., $\hat{f}(\vec{x}) = \sum_{j=1}^{p} \alpha_j \psi_j \left(\vec{\theta}_j, \vec{x}\right) + \alpha_0$. Let us re-write this expression in a matrix form $\hat{f}(\vec{x}) = \vec{\psi}(\Theta, \vec{x})\, \vec{\alpha}$, where we denote vectors by caps with vector signs and matrices by caps ($\vec{\alpha} = \{\alpha_j\}_{j=0}^{p}$, $\Theta = \{\vec{\theta}_j\}_{j=1}^{p}$). The row-vector $\vec{\psi}(\Theta, \vec{x})$ consists of dictionary functions values at the point $\vec{x}$ ($\psi_0 \equiv 1$ corresponds to α_0). Therefore the approximator $\hat{f}(\vec{x})$ is defined by the matrix Θ of the dictionary functions parameters and by the vector $\vec{\alpha}$ of the linear combination coefficients. In order to form the dictionary we use sigmoid functions (sigmoids):

$$\psi_j \left(\vec{\theta}_j, \vec{x}\right) = \sigma\left(\vec{x}^{\mathrm{T}} \theta_j + \theta_j^0\right),\ \vec{\theta}_j = (\theta_j, \theta_j^0),\ \theta_j \in R^d,\ \theta_j^0 \in R^1, \text{ where } \sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},\ z \in R^1.$$

The dictionary consisting of such functions can be used for approximation of rather wide class of functions f, see the results in the papers [5, 6]. For example in [5] it is shown that the approximator $\hat{f}$, composed of p sigmoid functions, allow to get the approximation accuracy of the order $O\left(1/p^{\frac{1}{d}}\right)$. However if we include in the dictionary not arbitrary sigmoid functions but select them depending on the approximated function f ("tune" the dictionary functions), then the approximation accuracy has the order $O(1/p)$.

2.2 Structure of the Approximation Algorithm

In order to construct the dictionary we need to select type and number of functions in the dictionary and initialize their parameters. It is impossible to determine the dictionary functions parameters explicitly. We apply the standard approach for approximation construction, which uses partitioning of the original sample into two parts $S_{learning} = S_{train} \cup S_{validation}$ in order to prevent over-training [7].

Algorithm 1 (Main Steps of the Approximation Construction Algorithm)

1. *Model Selection Step (in the considered case the model structure is defined by the number p of the dictionary functions).*

2. *Initialization Step: a) divide the sample $S_{learning}$ into sub-samples S_{train} and $S_{validation}$, b) set the initial values of the dictionary functions parameters Θ and expansion coefficients $\vec{\alpha}$.*

3. Training Step: a) iteratively minimize $Q\left(S_{train}, \hat{f}\right)$ with respect to the parameters Θ and $\vec{\alpha}$, b) stop minimization if the error $Q\left(S_{validation}, \hat{f}\right)$ begin to increase.

Each of the steps of Algorithm 1 is a separate sub-problem. There exists a lot of methods for solution of these sub-problems. In this paper we propose more efficient methods (except a solution for the sub-problem of the dictionary size p selection). As for the choice of the dictionary size p then usually some upper bound on its value is defined depending on the learning sample size and the exact value of p is selected using cross-validation.

2.3 Optimization Algorithm

Usually in order to optimize the error function (see step 3 of algorithm 1) with respect to parameters of complex regression models (for example, multi-layer neural networks) gradient methods are used due to very high dimensionality of the parametric space. Since in our case the number of parameters is relatively small then we can use second order optimization methods. The Gauss-Newton method [8] is the most common method for minimizing functions of the form $Q\left(S_{train}, \hat{f}\right) = \left(\vec{y} - \hat{f}(X)\right)^{\mathrm{T}} \left(\vec{y} - \hat{f}(X)\right)$, where X is a matrix of all points from S_{train} and $\vec{y} = f(X)$ is a vector of the function f values at these points. In fact the main difference between this method and the Newton method consists in how the matrix of second order derivatives of the error function is calculated. Let us denote by $\Omega = \{\Theta, \vec{\alpha}\}$ the set of all parameters of the model (parameters of the dictionary functions and the corresponding expansion coefficients) then assuming that a residual vector components $\vec{e} = \hat{f}(X) - \vec{y}$ are small we get that

$$Q_{\Omega\Omega} = \hat{f}_{\Omega}^{\mathrm{T}} \hat{f}_{\Omega} + \sum_{i=1}^{N_{train}} e_i \hat{f}_{\Omega\Omega}\left(\vec{x}_i\right) \approx \hat{f}_{\Omega}^{\mathrm{T}} \hat{f}_{\Omega} = J^{\mathrm{T}} J, \tag{1}$$

where $J = \hat{f}_{\Omega}$ is a matrix of the model $\hat{f}$ derivatives with respect to Ω at the points S_{train}. Since $Q_{\Omega\Omega} \approx J^{\mathrm{T}} J \succeq 0$ then the approximate matrix of the second derivatives, calculated according to the formula (1), is always non-negative definite. This partially solves degeneracy problem of the Newton method being applied to non-convex functions.

Nevertheless we should note that during the minimization of the error function approximation of the Hessian $Q_{\Omega\Omega} \approx J^{\mathrm{T}} J$ can become degenerate and non-invertible.

The Levenberg–Marquardt algorithm [9] was developed to solve this degeneracy problem. The main idea is to add identity matrix with regularization multiplier to the approximate Hessian matrix when searching the step size according to the Gauss-Newton method, i.e.

$$\Omega^k = \Omega^{k-1} - \left(J^{\mathrm{T}} J + \mu \mathrm{I}\right)^{-1} Q_{\Omega}. \tag{2}$$

The parameter μ defines behaviour of the algorithm: for small μ the step size is close to the step size of the Gauss-Newton method, for big μ the step is done along the anti-gradient with the size approximately equal to $\frac{1}{\mu}$. Therefore we can always find such value of the parameter μ, which provides decrease of the error function. When training the model we will use the Levenberg–Marquardt algorithm (this method is also realized in MatLab for ANN training).

3 Initialization Step

In this section we propose algorithms for solving two sub-problems of the initialization step: extraction of the validation sample and initialization of the model parameters. Also in sub-section 3.3 we describe a methodology for an experimental comparison of approximation algorithms and provide results of this comparison.

3.1 Extraction of the Validation Sample

Let us consider the first sub-problem of the initialization step, i.e. decomposition of the initial sample $S_{learning}$ into two parts S_{train} and $S_{validation}$, where S_{train} is used for an iterative tuning of the model parameters and $S_{validation}$ is used to estimate a generalization ability (an estimate of the proximity between the original function and its approximation) and to stop the iterative tuning process when the over-training appears.

In order to estimate the generalization ability using $S_{validation}$ the set of points $X_{validation} = \{\vec{x} \in S_{validation}\}$ should be "uniformly" distributed among other points of the learning sample $X_{learning} = \{\vec{x} \in S_{learning}\}$. Standard algorithms for approximation construction perform decomposition into the validation $S_{validation}$ and the training S_{train} samples randomly, which often results in inconsistent decomposition.

We will estimate the uniformity of $X_{learning}$ decomposition into the sets X_{train} and $X_{validation}$ using the following criterion

$$(3)\qquad U(X_{train}, X_{validation}) = \frac{1}{\#r^1} \sum_{\{\vec{x}_i, \vec{x}_j\} \in r^1} \frac{1}{\|\vec{x}_i - \vec{x}_j\|} - \frac{1}{\#r^2} \sum_{\{\vec{x}_i, \vec{x}_j\} \in r^2} \frac{1}{\|\vec{x}_i - \vec{x}_j\|},$$

where r^1 is the set of pairs of points such that both of points belongs either to X_{train}, or to $X_{validation}$; r^2 is the set of pairs of points such that one of them belongs to X_{train}, and another belongs to $X_{validation}$; $\#r^i$ is the cardinality of r^i, $i = 1, 2$. When minimizing $U(X_{train}, X_{validation})$ (with respect to different decompositions of $X_{learning}$) the distance between points from one and the same class is maximized and the distance between points from different classes is minimized that fully meets our objectives.

Optimization of the criterion (3) is an NP-hard combinatorial problem that can not be solved for reasonable time when the sample size $N_{learning} \gg 1$. On the other hand for the case $N_{learning} \sim 10$ we can perform optimization by the full search. Therefore let us consider the simplification of this optimization problem: we divide all the design domain into rather small hypercubes and we optimize the criterion locally by relocating points from one class (S_{train}) to another class ($S_{validation}$) only within each of the hypercubes.

In order to divide the design domain we use Classification And Regression Trees [10]. This method is based on the sequence of simple cuts of the design domain with respect to the input vector components. In each of the hypercubes, obtained during the previous iteration, we construct a constant approximation by averaging the output values of the points belonging to this hypercube. The input component and the location of the next cut are selected optimally in the sense of the mean square error of the corresponding piecewise constant approximation. There are a lot of criteria for stopping the tree construction process. These criteria are based on the generalization ability estimation of the tree [3]. Here as a stopping criterion we use an upper bound on the number of points belonging to each leave of the tree since the optimization complexity of the criterion (3) depends on this upper bound.

Algorithm 2 (Sample $S_{learning}$ decomposition)

1. *Construct a regression tree [10] with an upper bound on the number of points belonging to the tree leaves (the number of points should be bigger than $leaf_{min} = 8$ and smaller than $leaf_{max} = 16$) approximating the function f with piece-wise constant approximation. Let the number of leaves of the constructed tree is equal to some K.*

2. *Let s_k be a set of points $\vec{x} \in X_{learning}$ belonging to the leave with the number k.*

3. *For all $k = 1, \ldots, K$ using greedy algorithm (local optimization in each separate hypercube) we decompose the set s_k into subsets s_k^{val} and s_k^{tr} such that the criterion $U(s_k^{val}, s_k^{tr})$ takes its minimal value under the restriction that the fraction of the validation sample size is not smaller than $val_{part} = 0.2$.*

4. *Construct $S_{validation}$ and S_{train}*

$$S_{validation} = \left\{ \{\vec{x}, y = f(\vec{x})\} : \vec{x} \in \bigcup_{k=1}^{K} s_k^{val} \right\}, \quad S_{train} = \left\{ \{\vec{x}, y = f(\vec{x})\} : \vec{x} \in \bigcup_{k=1}^{K} s_k^{tr} \right\}.$$

The proposed algorithm contains two computationally expensive steps: construction of the regression tree (complexity is equal to $O(N_{learning} \log(N_{learning})))$ and local optimization of the criterion $U(s_k^{val}, s_k^{tr})$ (complexity is equal to $O(K) = O(N_{learning})$). For the second step the constant in $O(K)$ depends on the number of ways to decompose a leave with $leaf_{num} \in [leaf_{min}, leaf_{max}]$ points into two parts. Due to the restriction on the proportion between the sizes of the validation and the training samples this number is equal to $C_{leaf_{num}}^{[leaf_{num} \cdot val_{part}]}$, i.e. it is bigger than 28 (for $leaf_{num} = leaf_{min} = 8$) and smaller than 560 (for $leaf_{num} = leaf_{min} = 16$).

3.2 Initialization of the Model Parameters

Initialization of the dictionary functions parameters Θ significantly influences on the approximation construction process and the final approximation accuracy. The random methods Nguyen–Widrow (NW) [11] and SCAWI [12] are the most widely used algorithms for Θ initialization. These methods use some matrix of independent random variables multiplied by a fixed factor defining scale and smoothness of the dictionary functions.

Note that a random vectors generation in high-dimensional spaces leads to their clustering thus giving rise to clustering of the directions $\{\theta_j\}_{j=1}^{p}$. We propose an initialization algorithm, which generates a rich functional dictionary (in terms of a uniform distribution of the directions). We generate the directions $\{\theta_j\}_{j=1}^{p}$ uniformly on the unit sphere using a method from [13]. This method is based on a normalization of points generated by a multivariate normal distribution. The method uses the invariance property of a normal density relative to an arbitrary rotation and efficiently generates points with uniform distribution on the unit sphere.

Let us now consider how to select norms values (defined by some scaling multipliers) of the vectors $\{\theta_j\}_{j=1}^{p}$. The value of the norm influences the smoothness of the corresponding sigmoid: for a sufficiently small value the sigmoid is almost linear on the compact D, for a big value the sigmoid behaves like the step function. If one and the same value is used for all norms then all dictionary functions have one and the same smoothness and the dictionary is not "rich" enough. Therefore in order to define the norms values we use multipliers generated by the uniform distribution in some range. Thus the vectors $\left\{\theta_j\right\}_{j=1}^p$ have different norms and finally the dictionary contains sigmoids with different smoothness.

Algorithm 3 (Initialization of the parameters Θ)

1. *Construct a matrix* S *of size* $p \times d$ *containing* p *vectors, generated by the uniform distribution on a* d*-dimensional sphere of unit radius.*

2. *Let* $r = \sqrt{d} \cdot p^{1/N_{train}}$ *and generate values of scaling multipliers* $\xi_j, j = 1, \ldots, p$ *uniformly randomly on* $[0, r]$. *Such definition of* r *guarantees that the number of points belonging to the domain of a sigmoid saturation is small [11].*

3. *Let us define the sigmoids direction vectors* $\left\{\theta_j\right\}_{j=1}^p$ *according to the formula* $\theta_j = \xi_j \cdot \mathrm{S}_j$, $j = 1, \ldots, p$, *where* S_j *is a* j*-th line of the matrix* S. *We define the offset values* $\left\{\theta_j^0\right\}_{j=1}^p$ *in the same way as in [11], i.e., according to the uniform grid on the interval* $[-r, r]$.

In the paper [16], written by authors, comparison of popular random initialization methods with the proposed method and several specially developed deterministic initialization methods can be found.

3.3 Experimental Results

The proposed algorithms for the initialization step should be compared with the standard approaches in terms of the average error of approximation and its variation. For generation of test problems we use a set of artificial multidimensional functions used for testing optimization algorithms. This choice is due to the fact that in the framework of surrogate modelling instead of original objective function (and/or constraints) their approximations are used. We distinguish two types of test problems:

1. Test functions with fixed dimension of the input vector $\vec{x}$ — allinit, beale, hartmann, ishigami, wbd.

2. Test functions with dimension of the input vector $\vec{x}$ that can be set to some predefined value — gSobol, michalewicz, rosenbrock, whitley, zdt3. In experiments we vary the input dimension from 3 to 10.

Exact formulas for the test functions can be found in [14], [15].

As a relative approximation error we use the root-mean-square error normed by the analogous error for the constant approximation:

$$E\left(S_{test}, \hat{f}\right) = \sqrt{\frac{\sum_{i=1}^{N_{test}} \left(y_i - \hat{f}\left(\vec{x}_i\right)\right)^2}{\sum_{i=1}^{N_{test}} \left(y_i - \bar{y}\right)^2}}, \ \bar{y} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} y_i. \tag{4}$$

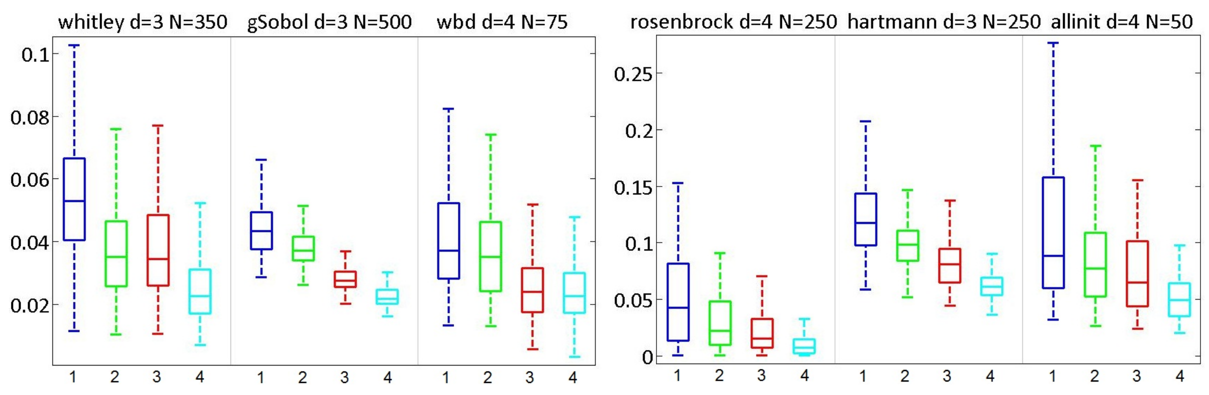

Figure 1: Method for solving problems of the initialization step. Errors for some tests

The error comparable with 1 corresponds to a very inaccurate approximation and the error comparable with 10^{-3} corresponds to a very accurate approximation. In order to calculate the error we use a separate test set S_{test}.

For each function (or pair "function-input dimension" for test problems of the second kind) we select the learning sample size such that the approximation error (4), obtained using a standard approximation method, belongs to the interval $0.01-0.2$ since this range is of great interest for practice (usually smaller values are not required in practice). By the standard approximation method we mean the realization of ANN from MatLab such that Nguyen–Widrow algorithm is used for the initialization, the validation sample is extracted randomly, and the training is performed by the Levenberg–Marquardt algorithm. In order to select the optimal dictionary size (the number of functions p) in each experiment we use brute force algorithm with the criterion defined by the approximation error, estimated using cross-validation. Each separate experiment (the test function, the input dimension, and the learning sample size are fixed) has the following set up: we generate 10 random learning sample $S_{learning}$; for each sample we construct 10 approximators. We compare the following algorithms: the standard ANN algorithm (the methods is denoted on plots by number 1), the standard ANN with the proposed method for the validation sample extraction (method 2), the standard ANN with the proposed method for parameters initialization (method 3), the standard ANN with both of the proposed methods for the initialization step (method 4).

In the most cases the proposed algorithms for the initialization step (extraction of the validation sample and initialization of the parameters), applied either independently or simultaneously, improve the final quality of the approximation models. It is interesting that the combination of the proposed algorithms (method 4) in most cases makes it possible to obtain the model with the superior accuracy compared with the accuracies of the models obtained using these algorithms independently. Therefore new algorithms are more effective, see examples of diagrams in figure 1, and also Dolan–More curves in section 5.

4 Approximation Training

4.1 Separability of Variables

Let us consider the error function on the training sample as a function of the parameters Θ and $\vec{\alpha}$:

$$Q\left(S_{train}, \hat{f}\right) = Q\left(\Theta, \vec{\alpha}\right) = \left(\vec{y} - \hat{f}\left(X, \Theta, \vec{\alpha}\right)\right)^{\mathrm{T}} \left(\vec{y} - \hat{f}\left(X, \Theta, \vec{\alpha}\right)\right).$$

We should note that the dependence of the error function Q on the dictionary functions parameters Θ is non-linear and very complex. At the same time the dependence of Q on expansion coefficients $\vec{\alpha}$ is quadratic. For the fixed Θ the optimal values of $\vec{\alpha}$ can be found by the least squares method:

$$\text{(5)} \quad \vec{\alpha}\left(\Theta\right) = \left(\Psi\left(\Theta\right)^{\mathrm{T}} \Psi\left(\Theta\right)\right)^{-1} \Psi\left(\Theta\right)^{\mathrm{T}} \vec{y}, \text{ where } \Psi\left(\Theta\right) = \left\{\vec{\psi}\left(\Theta, \vec{x}_i\right), i = 1, \ldots, N_{train}\right\}.$$

We take this fact into account when tuning parameters of the approximator. For this we consider the objective function $R\left(\Theta\right) = Q\left(\Theta, \vec{\alpha}\left(\Theta\right)\right)$, where $\vec{\alpha}\left(\Theta\right)$ is calculated according to formula (5). When calculating the expansion coefficient according to (5) a non-linear dependence $\vec{\alpha} = \vec{\alpha}\left(\Theta\right)$ should be taken into account when calculating the derivatives R_Θ and $R_{\Theta\Theta}$. An algorithm for such calculations was proposed in [17] and its theoretical properties were investigated in [18].

However this algorithm has significant shortcoming: in formula (5) we use inversion of the matrix $\Psi^{\mathrm{T}}\Psi$, which in general can be degenerate. In such case the inverse matrix does not exists and it is not reasonable to use the error function $R\left(\Theta\right)$. Such kind of situations are investigated in details in the framework of the linear regression methods [19]. It can be shown that even if the matrix $\Psi^{\mathrm{T}}\Psi$ is not degenerate but is ill-conditioned than estimates of the coefficients $\vec{\alpha}$ are unstable (in statistical terms this means that the estimate of $\vec{\alpha}$ has very big variance). In such case calculated values of the gradient and the Hessian of the error function are also unstable: even with small variations of the parameters Θ the first and the second derivatives of the error function change significantly resulting in a very low training speed.

Let us consider a classical approach for regularization in linear regression problems, namely, ridge regression [19]. We add to the matrix $\Psi^{\mathrm{T}}\Psi$ a positively definite matrix λI_p, where I_p is a unit matrix with sizes $p \times p$ and $\lambda > 0$ is a some constant. In such case formula (5) takes the form

$$\text{(6)} \qquad \vec{\alpha}\left(\Theta\right) = \left(\Psi\left(\Theta\right)^{\mathrm{T}} \Psi\left(\Theta\right) + \lambda I_p\right)^{-1} \Psi\left(\Theta\right)^{\mathrm{T}} \vec{y}.$$

It is obvious that we can reach any level of matrix $\left(\Psi\left(\Theta\right)^{\mathrm{T}} \Psi\left(\Theta\right) + \lambda I_p\right)$ conditionality by increasing the value of λ. Using the ridge regression is equivalent to re-definition of the error function in the following way:

$$\widetilde{R}\left(\Theta\right) = \widetilde{Q}\left(\Theta, \vec{\alpha}\left(\Theta\right)\right) = \left(\vec{y} - \hat{f}\left(X, \Theta, \vec{\alpha}\left(\Theta\right)\right)\right)^{\mathrm{T}} \left(\vec{y} - \hat{f}\left(X, \Theta, \vec{\alpha}\left(\Theta\right)\right)\right) + \lambda \vec{\alpha}\left(\Theta\right)^{\mathrm{T}} \vec{\alpha}\left(\Theta\right).$$

Statement 1 *The gradient and the Hessian of the error function $\widetilde{R}$ can be calculated according to the following formulas:* $\widetilde{R}_\Theta = \widetilde{Q}_\Theta = \vec{e}^{\mathrm{T}}\mathrm{J}$,

$$\text{(7)} \qquad \widetilde{R}_{\Theta\Theta} = \mathrm{J}^{\mathrm{T}}\mathrm{J} + \vec{e} \odot \hat{f}_{\Theta\Theta} - \left(\vec{e}^{\mathrm{T}} \odot \Psi_\Theta + \mathrm{J}^{\mathrm{T}}\Psi\right) \left(\Psi^{\mathrm{T}}\Psi + \lambda \mathrm{I}\right)^{-1} \left(\vec{e}^{\mathrm{T}} \odot \Psi_\Theta + \mathrm{J}^{\mathrm{T}}\Psi\right)^{\mathrm{T}}.$$

Proof. Let us find the first derivatives of the function $\widetilde{R}(\Theta)$:

$$\widetilde{R}_\Theta = \widetilde{Q}_\Theta + \widetilde{Q}_\alpha \alpha_\Theta = \widetilde{Q}_\Theta + \vec{0}\alpha_\Theta = \widetilde{Q}_\Theta. \tag{8}$$

$\widetilde{Q}_\alpha = \vec{0}$ since coefficients $\vec{\alpha}(\Theta)$, obtained using (6), is the minimizer of the function $\widetilde{Q}(\Theta, \vec{\alpha}(\Theta))$. Let us differentiate equality (8) with respect to Θ:

$$\widetilde{R}_{\Theta\Theta} = \widetilde{Q}_{\Theta\Theta} + \widetilde{Q}_{\Theta\alpha}\alpha_\Theta. \tag{9}$$

Contrary to formula (8) the second summand is not equal to 0. Therefore it is necessary to calculate α_Θ. Taking into account the equality $\widetilde{Q}_\alpha = \vec{0}$ as a consequence we get that the differential $d\widetilde{Q}_\alpha = 0$:

$$d\widetilde{Q}_\alpha = \widetilde{Q}_{\alpha\alpha} d\alpha + \widetilde{Q}_{\alpha\Theta} d\Theta = 0 \quad \Rightarrow \quad \alpha_\Theta = \frac{d\alpha}{d\Theta} = -\widetilde{Q}_{\alpha\alpha}^{-1}\widetilde{Q}_{\alpha\Theta}.$$

Then formula (9) takes the form $\widetilde{R}_{\Theta\Theta} = \widetilde{Q}_{\Theta\Theta} - \widetilde{Q}_{\Theta\alpha}\widetilde{Q}_{\alpha\alpha}^{-1}\widetilde{Q}_{\alpha\Theta}$.

Now let us write explicitly expressions for the derivatives of the function $\widetilde{Q}$. Let us denote by $\mathrm{J} = \hat{f}_\Theta(X)$ the matrix of the derivatives of the model $\hat{f}$ with respect to Θ at the points S_{train}. Therefore

$$\widetilde{Q}_\Theta = \vec{e}^{\mathrm{T}}\mathrm{J},\ \widetilde{Q}_{\Theta\Theta} = \mathrm{J}^{\mathrm{T}}\mathrm{J} + \vec{e} \odot \hat{f}_{\Theta\Theta},\ \widetilde{Q}_{\alpha\alpha} = \Psi^{\mathrm{T}}\Psi + \lambda \mathrm{I},\ \widetilde{Q}_{\Theta\alpha} = \vec{e}^{\mathrm{T}} \odot \Psi_\Theta + \mathrm{J}^{\mathrm{T}}\Psi. \tag{10}$$

Final formula (7) can be obtained directly from $\widetilde{R}_{\Theta\Theta} = \widetilde{Q}_{\Theta\Theta} - \widetilde{Q}_{\Theta\alpha}\widetilde{Q}_{\alpha\alpha}^{-1}\widetilde{Q}_{\alpha\Theta}$ using expressions for the derivatives from (10). ■

Assuming the residual vector $\vec{e}$ to be small we neglect in (7) all terms with $\vec{e}$:

$$\widetilde{R}_{\Theta\Theta} \approx \mathrm{J}^{\mathrm{T}}\mathrm{J} - \left(\mathrm{J}^{\mathrm{T}}\Psi\right)\left(\Psi^{\mathrm{T}}\Psi + \lambda\mathrm{I}\right)^{-1}\left(\mathrm{J}^{\mathrm{T}}\Psi\right)^{\mathrm{T}}. \tag{11}$$

Therefore we get explicit formulas for calculation of the gradient and the approximate Hessian matrix of the modified error function $\widetilde{R}(\Theta)$ that are necessary for optimization based on the Levenberg–Marquardt algorithm, see formulas (1) and (2).

We should note that the product $\mathrm{J}^{\mathrm{T}}\mathrm{J} \approx \widetilde{Q}_{\Theta\Theta}$ is non-negatively definite. Let us show that the Hessian $\widetilde{R}_{\Theta\Theta}$ also has this property. In such case we can construct accurate local-quadratic approximations of the error function during its optimization.

Statement 2 *The matrix* $\mathrm{H} \overset{\text{def}}{=} \mathrm{J}^{\mathrm{T}}\mathrm{J} - \left(\mathrm{J}^{\mathrm{T}}\Psi\right)\left(\Psi^{\mathrm{T}}\Psi + \lambda\mathrm{I}\right)^{-1}\left(\mathrm{J}^{\mathrm{T}}\Psi\right)^{\mathrm{T}}$ *is non-negatively definite for any* $\lambda \geq 0$.

Proof. Let us rewrite matrix H in the following form:

$$\mathrm{H} = \mathrm{J}^{\mathrm{T}}\mathrm{J} - \left(\mathrm{J}^{\mathrm{T}}\Psi\right)\left(\Psi^{\mathrm{T}}\Psi + \lambda\mathrm{I}_p\right)^{-1}\left(\mathrm{J}^{\mathrm{T}}\Psi\right)^{\mathrm{T}} = \mathrm{J}^{\mathrm{T}}\left(\mathrm{I}_N - \Psi\left(\Psi^{\mathrm{T}}\Psi + \lambda\mathrm{I}_p\right)^{-1}\Psi^{\mathrm{T}}\right)\mathrm{J}.$$

Let $\Psi = \mathrm{VQU}^{\mathrm{T}}$ be a singular value decomposition for the matrix of regressors then

$$\begin{aligned}\Psi\left(\Psi^{\mathrm{T}}\Psi + \lambda\mathrm{I}_p\right)^{-1}\Psi^{\mathrm{T}} =& \mathrm{VQU}^{\mathrm{T}}\left(\mathrm{UQ}^2\mathrm{U}^{\mathrm{T}} + \lambda\mathrm{UU}^{\mathrm{T}}\right)^{-1}\mathrm{UQV}^{\mathrm{T}} = \\ =& \mathrm{VQU}^{\mathrm{T}}\mathrm{U}\left(\mathrm{Q}^2 + \lambda\mathrm{I}_p\right)^{-1}\mathrm{U}^{\mathrm{T}}\mathrm{UQV}^{\mathrm{T}} = \mathrm{VQ}^2\left(\mathrm{Q}^2 + \lambda\mathrm{I}_p\right)^{-1}\mathrm{V}^{\mathrm{T}}.\end{aligned}$$

Let $\tilde{q}_j$ be eigenvalues of the matrix $\mathrm{V}\mathrm{Q}^2\left(\mathrm{Q}^2+\lambda\mathrm{I}_p\right)^{-1}\mathrm{V}^{\mathrm{T}}$ then $\tilde{q}_j = \frac{q_j^2}{q_j^2+\lambda}$, where q_j are elements of the matrix Q. The eigenvalues of the matrix $\mathrm{P} = \mathrm{I}_N - \Psi\left(\Psi^{\mathrm{T}}\Psi+\lambda\mathrm{I}_p\right)^{-1}\Psi^{\mathrm{T}}$ are not smaller than the difference of the minimal eigenvalue of I_N and the maximal eigenvalue from the set $\tilde{q}_j$. It is obvious that this difference is non-negative since $(1-\max_j \tilde{q}_j) = \left(1-\max_j \frac{q_j^2}{q_j^2+\lambda}\right) \geq 0$ for $\lambda \geq 0$. Therefore the matrix P is non-negatively definite and the matrix $\mathrm{H} = \mathrm{J}^{\mathrm{T}}\mathrm{P}\mathrm{J}$ is also non-negatively definite. ■

Now let us estimate the computational complexity of formula (11) for the Hessian calculation $\widetilde{R}_{\Theta\Theta}$. The size of the Jacobian matrix J is equal to $N_{train} \times p(d+1)$, the size of the matrix with regressors Ψ is equal to $N_{train} \times p$. We need to perform $O(N_{train}p^2d^2)$ operations in order to calculate the main term $\mathrm{J}^{\mathrm{T}}\mathrm{J}$, which is necessary also for calculation of the standard error function. At the same time additional summand $\left(\mathrm{J}^{\mathrm{T}}\Psi\right)\left(\Psi^{\mathrm{T}}\Psi+\lambda\mathrm{I}\right)^{-1}\left(\mathrm{J}^{\mathrm{T}}\Psi\right)^{\mathrm{T}}$ can be calculated for $O(N_{train}p^2d+p^3+p^3d+p^3d^2) = O(N_{train}p^2d+p^3d^2)$ operations. Since in the considered class of approximation problems $N_{train} \gg d$, $N_{train} \gg p$ then calculation of the additional summand requires significantly smaller number of operations compared to the number of operations for calculation of the main summand.

The proposed error function $\widetilde{R}(\Theta)$ not only increases approximation accuracy but also decreases training time compared to the training time when using the standard error function $Q(\Theta, \vec{\alpha})$ [20]. This advantage can be explained by two reasons: some part of the parameters are estimated optimally on each iteration of the training algorithm and adaptive regularization increases numerical stability of the training process. In this work we assume that the output y dimension is equal to 1, but for many applied problems the output y can be multidimensional and its dimension d_y can even be higher than the input dimension d. In such case the number of parameters of the standard error function is $\frac{d+d_y}{d}$ times higher than that of the modified error function. Thus separability of the variables can significantly decrease the number of optimized parameters in some problems.

4.2 Adaptive Regularization

Standard approaches for regularization of models with the structure $\hat{f}(\vec{x}) = \vec{\psi}(\Theta, \vec{x})\vec{\alpha}$ use the L_2 penalty on all parameters of the model [3] and a regularization coefficient is determined experimentally using some additionally extracted validation samples and multiple training of the surrogate model. In [21] some Bayesian approach is proposed for the regularization parameter selection and again the L_2 penalty on all parameters of the model are used. This method has two key shortcomings: selection of the regularization parameter does not take into account the error of approximation; the penalty incorporates norms of the parameters Θ and $\vec{\alpha}$ with equal weights and does not take into account essentially different nature of these parameters.

In the proposed approach we penalize only the expansion coefficients $\vec{\alpha}$ and regularization parameter λ is selected optimally (in some sense) with taking into account the error of approximation.

When tuning the regularization parameter λ during the training process the functional dependency $\lambda = \lambda(\Theta)$ appears. In general case we should take into account this dependency when calculating the derivatives of the error function with respect to Θ. However in the framework of the considered approximation problem optimization of the error func-

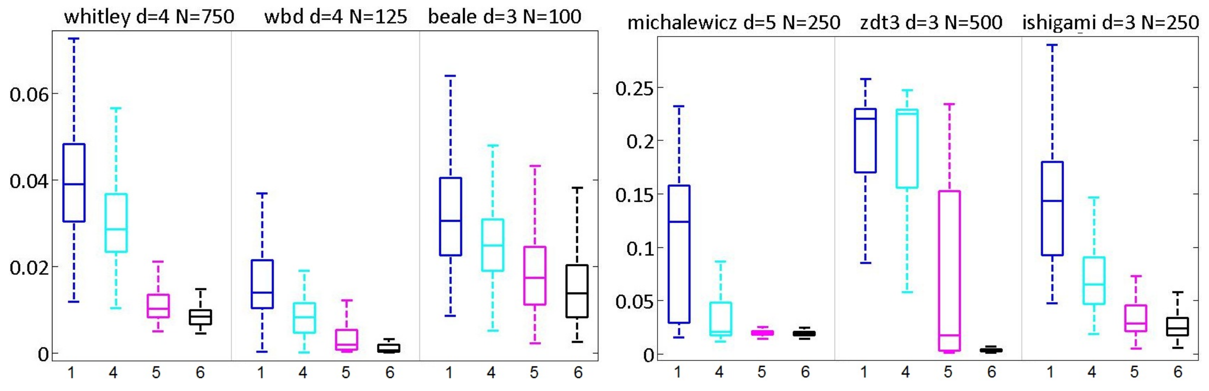

Figure 2: Methods for solving problems of the training step. Errors for some tests.

tion on the set S_{train} can be considered as the process for generation of different models with the final aim to obtain the model with the smallest error on the independent test set S_{test} rather than for finding the minimum of the error function on the train set S_{train}. Due to this remark change of the regularization parameter value during the training process is allowed and we can neglect these changes when calculating the derivatives of the error function.

In order to select the regularization parameter λ on each iteration of the training algorithm we minimize the GCV criterion (Generalized Cross Validation [3]) estimating the approximation error on the test sample in linear regression problems ($\vec{\alpha}(\lambda)$ is calculated according to (6))

$$GCV(\Psi,\lambda) = \frac{\sum_{i=1}^{N_{train}} \left(y_i - \hat{f}(x_i)\right)^2}{\left(1 - \frac{1}{N_{train}}\mathrm{tr}\,(\mathrm{L})\right)^2} = \frac{\left(\vec{y} - \Psi\vec{\alpha}(\lambda)\right)^{\mathrm{T}}\left(\vec{y} - \Psi\vec{\alpha}(\lambda)\right)}{\left(1 - \frac{1}{N_{train}}\mathrm{tr}\left(\left(\vec{\Psi}^{\mathrm{T}}\vec{\Psi} + \lambda \mathrm{I}\right)^{-1}\vec{\Psi}^{\mathrm{T}}\vec{\Psi}\right)\right)^2}.$$

Let us note that when minimizing the criterion GCV with respect to λ it is necessary to control condition number of the matrix $\Psi^{\mathrm{T}}\Psi + \lambda I$ in order the training process to be stable [22]. It is proposed to impose a lower bound on the value of λ such that provides necessary level of the conditionality (10^{12} in our realization of the algorithm).

4.3 Experimental Results

Let us use the testing methodology described in section 3.3. We compare the following methods: the standard method (method 1), the method incorporating algorithms from sections 3.1 and 3.2 (method 4), the method incorporating only the modified error function (method 5), the method incorporating all the proposed algorithms (method 6). The most indicative results are given on the figure 2. The proposed modification of the error function allows to significantly improve approximation quality for some problems (for example, function michalewicz). If we consider method 6, which incorporates all the proposed algorithms, then we can see that this method not only additionally increases approximation accuracy compared to the best of two methods 4 and 5, but also significantly decreases the approximation error in some cases (for example, zdt3 function).

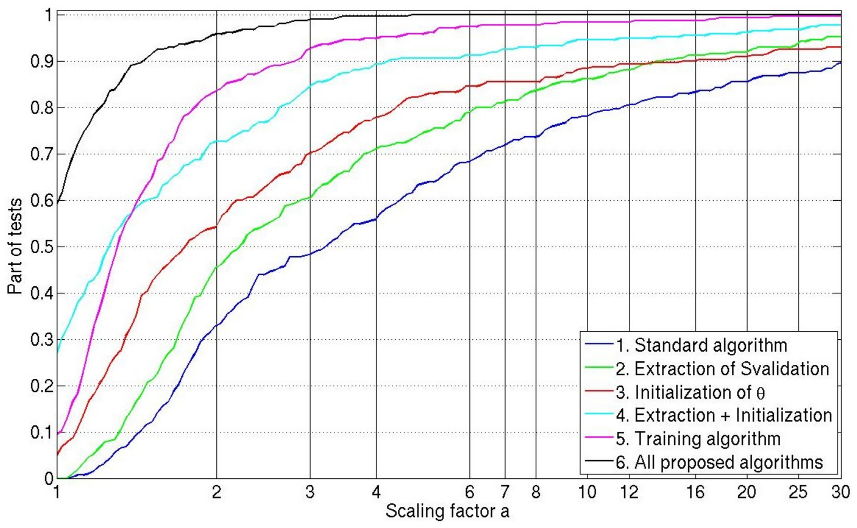

Figure 3: Results for artificial problems. Median values (accuracy)

5 Experimental Results

In this section we show results of full experiments on artificial functions and compare the proposed approach with other similar approaches on some applied problems of surrogate modelling. We use Dolan–More curves $P_k(a)$ [23] for visualization of the results. The quantity $P_k(a)$ shows on which fraction of the problems the errors of the considered approximation method k is not $a \geq 1$ higher than the minimal (among all the considered methods) approximation error for the corresponding approximation problems.

5.1 Artificial Functions

Using Dolan–More curves let us compare the standard algorithm with all algorithms from sub-sections 3.3 and 4.3 on all problems used for testing. As a “separate problem” we consider all independently generated samples, i.e. 10 problems correspond to each function. We use the following criteria of approximation quality: median of the errors (accuracy) and standard deviation of the errors (scatter of the errors). For each problem we run each method 10 times (the error for each run is estimated using formula (4)) and estimate the accuracy and the scatter using results of these runs. Results of comparison are given on figures 3 and 4. We can see that the main contribution into accuracy is obtained due to the modified error function with the adaptive regularization, but the algorithms of the initialization step also significantly improve the accuracy of the methods (see curves for methods 5 and 6).

5.2 Real Applied Problems

Let us compare approximation quality of the proposed method (combining all the proposed algorithms) with widely-used methods for approximation construction. We use realization of such methods in MatLab and modeFrontier. This software systems are used by many of industrial companies. There are a number of methods for approximation construction, implemented in MatLab and modeFrontier, namely, ANN, regression based

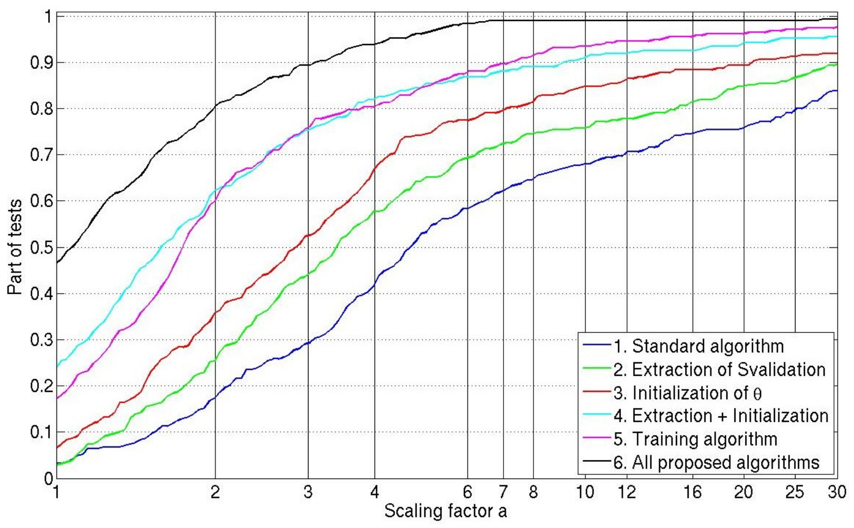

Figure 4: Results for artificial problems. Standard deviation (scatter)

on Gaussian Process, etc. We should note that many of these methods have serious restrictions on possible characteristics of the sample (input dimension d and sample size $N_{learning}$), which in turn limits the applicability of the methods and as a consequence reduces the accuracy of approximation. In most of the cases such restrictions are due to algorithmic peculiarities of the corresponding realizations.

Let us consider results on some indicative problems covering a wide range of sample sizes and input dimensions:

- the strength of the composite structure of the aircraft fuselage [4],
- aerodynamic characteristics of the aircraft wing [24],
- structure of the sand in the field [25],
- concrete compressive strength [26].

Reference results (of approximation algorithms from MatLab and modeFrontier) were obtained in 2010 during work on the PhD thesis. We measure the approximation quality using error (4), calculated using the independent test set S_{test}. Results of the comparison are given in table 1 (for the problem of approximation of the aircraft wing aerodynamic characteristics some of the methods do not work due to high input dimensionality).

6 Conclusions

We consider problem of approximation of a multidimensional dependency based on a linear expansion in a dictionary of parametric functions. We propose new methods for solving sub-problems that arise in the framework of approximation construction problem. Each of these methods as well as their combination significantly increases the approximation quality compared to the approximation quality obtained using standard methods. Using the proposed algorithm we were able to solve important applied problems, see for example [4].

Table 1: Relative Approximation Error

Problem		Composite Struct.	Wing Charact.	Sand	Concrete
Dimension of $\vec{x}$		16	78	3	8
Sample size		50000	65000	10000	1030
	Proposed approach	0,092	0,159	0,091	0,320
MatLab	Lin. Reg.	0,616	0,330	0,698	0,6391
	Quad. Reg.	0,390	-	0,608	0,485
	ANN	0,194	0,258	0,132	0,350
	RBF	0,336	0,628	0,464	0,371
mode Frontier	K-NN	0,597	1,115	0,470	0,529
	Anisotr. Kriging	0,528	-	0,813	2,521
	Kriging	0,382	1,031	0,937	0,889
	RBF	0,363	9,392	0,494	0,367
	ANN	0,299	1,424	0,356	0,730
	Gaussian Proc.	0,807	-	0,561	2,088

References

[1] *Forrester A., Sobester A., Keane A.* Engineering Design via Surrogate Modelling. A Practical Guide. Wiley. 2008.

[2] *Kuleshov A., Bernstein A.* Cognitive technologies in adaptive models of complex plants // Keynote papers of 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM'09). 2009. P. 70–81.

[3] *Hastie T., Tibshirani R., Friedman J.* The elements of statistical learning: data mining, inference, and prediction. Springer. 2008.

[4] *Grihon S., Alestra S., Burnaev E., Prikhodko P.* Optimization of Composite Structure based on Surrogate Modelling of Buckling Analysis // Proc. of the "Information Technologies and Systems" conf. 2012. P. 41–47.

[5] *Pinkus A.* Approximation theory of the MLP model in neural networks. Acta Numerica (8). Cambridge Univ. Press. Cambridge. 1999. P. 143–195.

[6] *P. Petrushev*, Approximation by ridge functions and neural networks. SIAM J. Math. Anal. 1998. V. 30. P. 155–189.

[7] *Vapnik V. N., Chervonenkis A. Ja.* Ordered Risk Minimization (I and II). Autom. Remote Control. 1974. V. 34. P. 1226–1235; 1974. V. 34. P. 1403–1412.

[8] *Nocedal J., Wright S.* Numerical Optimization, 2nd Edition, Springer. 2006. P. 664.

[9] *Marquardt D. W.* An Algorithm for Least-Squares Estimation of Nonlinear Parameters. Journal of SIAM. 1963. V. 11 No. 2. P. 431–441.

[10] *Breiman L.* Classification and regression trees. Wadsworth. 1984.

[11] *Nguyen D., Widrow B.* Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights // IJCNN International Joint Conference. 1990. P. 21–26.

[12] *Drago G. and Ridella S.* Statistically controlled activation weight initialization (SCAWI). Trans. Neur. Netw. IEEE Press. 1992. V. 3. No. 4. P. 627–631.

[13] *Rubinstein R. Y.* Generating random vectors uniformly distributed inside and on the surface of different regions. European Journal of Operational Research. 1982. V. 10. No. 2. P. 205–209.

[14] *Hedar A. R.* Global optimization test problems // http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm

[15] *Molga M., Smutnicki C.* Test functions for optimization needs // http://zsd.ict.pwr.wroc.pl/files/docs/functions.pdf

[16] *Belyaev M. G., Burnaev E. V., Erofeev P. D., Prikhodko P. V.* Comparison of the efficiency of the initialization methods for non-linear regression models // Proc. of the “Information Technologies and Systems” conf. 2011. P. 315–320.

[17] *Golub G. H., Pereyra V.* The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate. SIAM J. Numer. Anal. 1973 V. 10. P. 413–432.

[18] *Ruhe A., Wedin P. A.* Algorithms for separable nonlinear least squares problems. SIAM Review, 1980. V. 22. No. 3. P. 318–337.

[19] *Demidenko E. Z.* Linear and non-linear regression. Finance and stochastics. 1981.

[20] *Belyaev M. G., Lyubin A. D.* Peculiarities of the optimization problem, which arises when constructing an approximation of a multidimensional function // Proc. of the “Information Technologies and Systems” conf. 2011. P. 415–422.

[21] *Foresee D. and Hagan M.* Gauss-Newton approximation to Bayesian learning // Proc. of the Inter. conf. on Neural Networks. 1997. V. 3. P. 1930–1935.

[22] *Belyaev M. G., Burnaev E. V.* Adaptive regularization in the problem of multidimensional functions approximation // Proc. of the “Information Technologies and Systems” conf. 2009. P. 431–435.

[23] *Dolan E., More J.* Benchmarking optimization software with performance profiles. Mathematical Programming, Ser. A 91. 2002. P. 201–213.

[24] *Chervonenkis A. Ya., Chernova S. S., Zykova T. V.* Applications of kernel ridge estimation to the problem of computing the aerodynamical characteristics of a passenger plane (in comparison with results obtained with artificial neural networks). Automation and Remote Control. 2011. V. 72. No. 5. P. 1061–1067.

[25] IC Fault dataset // http://imperial.ac.uk/earthscienceandengineering/research/perm/icfaultmodel

[26] Concrete Compressive Strength dataset // http://archive.ics.uci.edu/ml/datasets