EasyChair Preprint

№ 1964

Solution of Partial Differential Equations on Radial Basis Functions Networks

Mohie Alqezweeni and Vladimir Gorbachenko

November 16, 2019

Solution of partial differential equations on radial basis functions networks

Mohie M. Alqezweeni[0000-0003-3036-8992] and Vladimir I. Gorbachenko[0000-0002-1012-8855]

Penza State University, 40, Krasnaya street, 440026 Penza, Russia
mohieit@mail.ru, gorvi@mail.ru

Abstract. The solution of boundary value problems described by partial differential equations on networks of radial basis functions is considered. An analysis of gradient learning algorithms for radial basis functions networks showed that the widely used first-order method, the gradient descent method, does not provide a high learning speed and solution accuracy. The fastest method of the second order - the trust region method is very complex. A learning algorithm based on the Levenberg-Marquardt method is proposed. The proposed algorithm, with a simpler implementation, showed comparable results in comparison with the trust region method.

Keywords: partial differential equations, radial basis functions networks, neural network learning, Levenberg-Marquardt method.

1 Introduction

In the modern industry, Digital Twin is widely used [1–2]. A digital twin is a dynamic virtual model of a system, process or service. A digital double is constantly learning and updating its parameters, receiving information from many sensors, correctly represents the state of a physical object. During learning, it uses current data from sensors, from control devices, from the external environment. Digital twins allows real-time monitoring of systems and processes and timely analysis of data to prevent problems before they occur, schedule preventative maintenance, reduce downtime, open up new business opportunities and plan future updates and new developments.

Digital doubles of objects with distributed parameters are mathematically boundary value problems for partial differential equations (PDE) [3]. In most cases, boundary value problems are solved by numerical methods, since analytical solutions exist only for a very limited range of problems. For the numerical solution of boundary value problems for PDE, the methods of finite differences and finite elements are widely used [4]. These methods require the construction of computational grids. Generating meshes for two and three-dimensional areas of complex configuration is a complex and time-consuming task. The complexity of grid formation for real problems often exceeds the complexity of solving a system of difference equations [5]. Large computational costs lead to the use of low-order approximations, which provide continuous approximation of the solution on the network, but not its partial derivatives. Modeling

of objects with distributed parameters by the methods of finite differences and finite elements is reduced to solving sparse systems of algebraic equations of very large dimension. These systems are characterized by poor conditioning, which requires high costs for their solution. Reconstructing a solution from its discrete approximation is a separate rather time-consuming task.

When modeling complex technical objects, software packages based on the finite element or finite difference method are usually used. However, modeling a real object with their help encounters a number of fundamental difficulties [6]. First, accurate information about differential equations describing the behavior of an object is usually absent due to the complexity of the description of the processes occurring in it. Secondly, to apply the methods of finite elements and finite differences, one needs to know the initial and boundary conditions, information about which is usually incomplete and inaccurate. Thirdly, during the operation of a real object, its properties and characteristics, parameters of the processes occurring in it can change. This requires appropriate adaptation of the model, which is difficult to carry out with models built on the basis of finite element methods and finite differences.

An alternative to finite difference methods and finite elements are meshless methods [7], most of which are projection methods. These methods give an approximate analytical solution in the form of a sum of basis functions multiplied by weights. As basis functions, radial basis functions (RBF) are popular [8–9]. Methods using RBF allow one to obtain a differentiable solution at an arbitrary point in the solution domain in the form of a function satisfying the required smoothness conditions, they are universal, allow working with complex geometry of computational domains, and are applicable for solving problems of any dimension. RBF-based methods require, for the selected parameters of the radial basis functions, to find the vector of weights, so that the resulting approximate solution ensures that the equation and boundary conditions are satisfied with an acceptable error on a certain set of sampling points. For example, the sum of the squared residuals at the sampling points should be small. The main disadvantage of using RBF is the need for unformalized selection of parameters of basis functions.

Promising is the implementation of meshless methods on neural networks. The solution of boundary value problems for PDE is possible on multilayer perceptrons [9–10]. But the most promising is the use of radial basis function networks (RBFN) [11], since RBFNs contain only two layers, one of which is linear, and the solution formation is local in nature, which simplifies the learning of such networks. The use of RBFN allows you to configure both weights and RBF parameters during learning networks. Applications of RBNF for solving boundary value problems are considered in the works of Jianyu L., Siwei L., Yingjian Q., Yaping H., Mai-Duy N., Tran-Cong T., Sarra S., Chen H., Kong L., Leng W., Kumar M., Yadav N., Vasilieva A.N., Tarkhova D.A., Gorbachenko V.I. [12–15].

To build digital models of twins, it is promising to use the ideas of machine learning and neural networks to build models of real objects. This approach allows you to build adaptive models that are refined and rebuilt in accordance with the observations of the object. Therefore, the urgent task is the development of neural network modeling technologies, a more complete account of historical and newly arriving data, im-

proving methods for automatically adjusting architecture and model parameters, classification and prediction methods [6]. Using neural network models allows us to develop a unified approach to solving various modeling problems. For example, in [16] a unified approach to solving direct and inverse boundary value problems described by partial differential equations was proposed.

The solution to the problem is formed in the learning process RBFN. Therefore, it is important to reduce network learning time. But at present, for learning RBFN in solving boundary value problems, mainly the simplest gradient methods of the first order based on gradient descent are used [10]. Second-order fast methods are practically not used in solving boundary value problems on RBFN. An exception is the confidence area method proposed in [15]. But the method is very complicated, since it requires at each iteration the solution of the minimization problem to solve the conditional minimization problem.

The aim of this work is to improve the algorithms for learning networks of radial basis functions in solving boundary value problems, which reduce the time of solving the problem.

2 Related works

RBF [8] are the functions of the distance of a space point from a function parameter called the center of the function: $\varphi\left(\left\|\mathbf{x}-\mathbf{c}\right\|,\mathbf{p}\right)$, where $\mathbf{x}$ — the space point, $\mathbf{p}$ — the vector of function parameters, $\mathbf{c}$ — the center of the radial basis function, $\left\|\mathbf{x}-\mathbf{c}\right\|$ — the Euclidean norm (distance) between the point and center. Various RBFs are applied. In this paper, we use the Gauss function (Gaussian)

$$\varphi(\|\mathbf{x}-\mathbf{c}\|,a)=\exp\left(-\frac{\|\mathbf{x}-\mathbf{c}\|^2}{2a^2}\right),$$

where $\mathbf{c}$ — the position of the function center, a — the shape parameter, often called the width.

When using RBF for solving boundary value problems, the type and parameters of RBF are selected before solving the problem. This procedure is informal, requires experimental verification and does not have unambiguous recommendations. Only some recommendations on choosing RBF and their parameters are known [17].

The solution of boundary value problems using RBF is based on the approximation of functions. Since when solving boundary value problems, an approximation of an unknown solution is performed, minimization of the residual at the sampling points is used. E. J. Kansa proposed a method for solving boundary value problems using RBF [18–19], which became the basis for other methods using RBF. We consider the boundary value problem in operator form

$$Lu(\mathbf{x})=f\left(\mathbf{x}\right),\ \mathbf{x}\in\Omega,\quad Bu(\mathbf{x})=p\left(\mathbf{x}\right),\ \mathbf{x}\in\partial\Omega, \tag{1}$$

where u — the solution to the problem; L — the differential operator; the operator B — the boundary condition operator; Ω — the solution domain; $\partial\Omega$ — the boundary of the region; f and p are known functions.

Inside the solution domain and at the boundary, many sampling points are defined

$$\{\mathbf{x}_i \mid_{i=1,2,\ldots,N} \subset \Omega\} \cup \{\mathbf{x}_i \mid_{i=N_1+1,N,\ldots,N+K} \subset \partial\Omega\}, \quad (2)$$

where N — the number of sampling points in the inner region of Ω, K — the number of sampling points on the border of $\partial\Omega$.

The solution to the problem is in the form of a weighted sum of basis functions

$$u_{\text{RBF}}(\mathbf{x}) = \sum_{j=1}^{M} w_j \varphi_j(\mathbf{x}), \qquad \mathbf{x} \in \bar{\Omega} = \Omega \cup \partial\Omega \quad (3)$$

where φ_j — RBF; w_j — weights, M — the number of RBF.

In (3), the number of RBFs is taken equal to the number of sampling points: $M = N + K$. RBF parameters are set. The unknown coefficients in (3) are found as a solution to a system of linear algebraic equations, which is obtained from the residuals of problem (1) at sampling points after substituting (3) in (1). For this, the RBF must be differentiable as many times as necessary. The result is a system of linear algebraic equations

$$\mathbf{A}\mathbf{w} = \mathbf{b}, \quad (4)$$

где $\mathbf{A} = \begin{bmatrix} \mathbf{G}_L \\ \mathbf{G}_B \end{bmatrix}$, $\mathbf{G}_L = \begin{bmatrix} L[\varphi_1(\mathbf{x}_1)] & L[\varphi_2(\mathbf{x}_1)] & L[\varphi_3(\mathbf{x}_1)] & \ldots & L[\varphi_N(\mathbf{x}_1)] \\ L[\varphi_1(\mathbf{x}_2)] & L[\varphi_2(\mathbf{x}_2)] & L[\varphi_3(\mathbf{x}_2)] & \cdots & L[\varphi_N(\mathbf{x}_2)] \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ L[\varphi_1(\mathbf{x}_N)] & L[\varphi_2(\mathbf{x}_N)] & L[\varphi_3(\mathbf{x}_N)] & \cdots & L[\varphi_N(\mathbf{x}_N)] \end{bmatrix}$,

$$\mathbf{G}_B = \begin{bmatrix} B[\varphi_1(\mathbf{x}_{N+1})] & B[\varphi_2(\mathbf{x}_{N+1})] & B[\varphi_3(\mathbf{x}_{N+1})] & \cdots & B[\varphi_N(\mathbf{x}_{N+1})] \\ B[\varphi_1(\mathbf{x}_{N+2})] & B[\varphi_2(\mathbf{x}_{N+2})] & B[\varphi_3(\mathbf{x}_{N+2})] & \cdots & B[\varphi_N(\mathbf{x}_{N+2})] \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ B[\varphi_1(\mathbf{x}_M)] & B[\varphi_2(\mathbf{x}_M)] & B[\varphi_3(\mathbf{x}_M)] & \cdots & B[\varphi_N(\mathbf{x}_M)] \end{bmatrix},$$

$$\mathbf{b} = \left[f(\mathbf{x}_1), f(\mathbf{x}_2), \quad \cdots \quad f(\mathbf{x}_{N_1}), \quad g(\mathbf{x}_{N_1+1}), \quad g(\mathbf{x}_{N_1+2}), \quad \cdots \quad g(\mathbf{x}_M) \right]^{\mathrm{T}},$$

$$\mathbf{a} = \left[w(\mathbf{x}_1), w(\mathbf{x}_2), \ldots, w(\mathbf{x}_M) \right]^{\mathrm{T}}.$$

System (4) has a square matrix and its solution is a weight vector $\mathbf{w}$. The Kansa method generates an asymmetric matrix, which makes it difficult to solve the system with a large number of sampling points. With a large number of sampling points, the $\mathbf{A}$ matrix is poorly conditioned. When using RBF with a global domain of definition, the matrix is dense, which also worsens conditioning. A serious drawback is the unformalized selection of the best RBF parameters

Known works do not consider the relationship between the number of RBF and the number of sampling points. Usually take the number of sampling points equal to the number of RBF. However, the ratio between the number of RBF M and the number of sampling points $N + K : M \propto (N + K)^{\frac{1}{3}}$, where $\propto$ means proportionality [20], is known for approximation problems. Since the number of sampling points in this case significantly exceeds the number of RBFs, system (4) is overridden. To solve such systems, the Singular Value Decomposition method is convenient [21].

When solving non-stationary problems, one can use RBF to approximate the differential operator with respect to spatial variables, preserving the differential operators with respect to time (direct method). The result is an ordinary differential equation containing a differential operator approximable by RBF. Более простым является прием, при котором производная по времени заменяется коечной разностью и на каждом временном слое с помощью RBF решается стационарная задача. A simpler method is when the time derivative is replaced by a bed difference and a stationary problem is solved on each time layer using RBF. For example, the equation $\frac{\partial u}{\partial t} = LU$ after approximating the time derivative takes the form $\frac{u^k - u^{k-1}}{\tau} = Lu^k$, where τ — the step of sampling time, k — the time layer number. Then, on the temporary k layer, the stationary $\tau Lu^k - u^k = -u^{k-1}$ problem is solved.

Thus, the use of RBF allows you to implement meshless methods and obtain a solution in an approximate analytical form. The resulting solution makes it possible to calculate the solution and its derivatives at arbitrary points in the region. But methods using RBF require solving poorly conditioned systems of linear algebraic equations with dense rectangular matrices. There are no formalized methods for determining the position and parameters of the RFB form. Networks of radial basis functions are free from most of these shortcomings, all parameters of which are determined during the networks learning.

RBFN includes two layers [11]. The first layer consists of RBFs that perform nonlinear transformation of the input vector $\mathbf{x} = [x_1, x_2, ..., x_d]$ — the coordinates of the point at which the approximation to the solution is calculated (d — the dimension of space). The second RBFN layer is a linear weighted adder

$$u(\mathbf{x}) = \sum_{m=1}^{M} w_m \varphi_m(\mathbf{x}; \mathbf{p}_m), \quad (5)$$

where M — the number of RBF, w_m — RBF weight φ_m, $\mathbf{p}_m$ — parameter vector.

The process of solving boundary value problems using RBFN was considerd using the example of problem (1) defined in the operator form. In the simplest case, it consists of 3 stages:

1. From the sets Ω and $\partial\Omega$ choose N internal and K boundary sampling points (2) (points at which the error of the solution is controlled). When there is no a priori information about the solution, it is advisable to use random uniform distribution of sampling points in the region and on the boundary of the solution. If there is a priori information about the solution of the problem, you can increase the number of sampling points in those areas in which it is necessary to obtain increased accuracy of the solution. For example, it is advisable to increase the number of sampling points in areas in which a change in the characteristics of the solution is expected.

Since the properties of the solution to the problem are a priori difficult to evaluate, you can first find a rough solution to the problem using the minimum number of sampling points, and then, having determined the areas in which the error functional takes on the greatest value, decide on the number of sampling points and their location. As already noted, the ratio between the number of RBF M and the number of sampling points $N + K$ is known. However, when approximating the solutions of boundary value problems using RBFN, this dependence gives an excessive number of sampling

points; therefore, it is necessary to select the number of sampling points. An increase in the number of sampling points leads to an increase in the computational complexity of the problem. Periodic random regeneration of a limited number of sampling points, used to prevent network retraining, reduces the number of sampling points.

2. Define the RBFN structure: network type, number of RBF, type RBF, set initial values for the vector of weights and parameter vectors of RBF. There are no definite recommendations for choosing the type of RBF. When solving a second-order PDE, it is necessary to calculate the second derivatives of the network output. Therefore, it is advisable to use the Gaussian function, the domain of definition of which is comparable with the domain of definition of its derivatives, which cannot be said of multiquads, for which there is a large spread of values. Unlimited values of multiquadrics also complicate their use in the uneven distribution of RBF centers. When choosing preliminary values, it is necessary to set the RBF parameters and the weight vector. The methods for choosing the location of the RBF centers are very similar to the methods for selecting sampling points. Centers can be arranged in nodes of a uniform grid or randomly. You can increase the density of RBF in areas where a change in the nature of the solution is expected. You can start the solution with a minimum amount of RBF and add RBF in areas with large error values during learning [12]. When placing RBF centers in the nodes of a uniform grid, it is advisable to set the same preliminary width values for all RBFs. The width values in this case are selected depending on the step size. With a random distribution of the centers, the width can be chosen randomly from a certain interval. The boundaries of the interval can be the same for all RBFs, or depend on the distance between the center of the RBF and the centers of its neighbors. Weights are usually triggered by small random numbers.

3. Perform network learning, ie select such values of weights and RBF parameters so that the error functional at the sampling points takes a minimum value. The solution of the boundary value problem (1) on RBFN is an approximation of an unknown solution on the set of sampling points (2). Since the solution at the sampling points is unknown, only minimization of the residuals on the set of sampling points is possible. To construct the functional error, the least squares method is used. The functional error for searching for $\mathbf{w}$ weights and $\mathbf{p}$ RBF parameters minimizing discrepancies at sampling points has the form

$$J(\mathbf{w},\mathbf{p}) = \sum_{i=1}^{N}\left[Lu_{\mathrm{RBF}}(\mathbf{x}_i;\mathbf{w},\mathbf{p}) - f(\mathbf{x}_i)\right]^2 + \lambda\sum_{i=N+1}^{N+K}\left[Bu_{\mathrm{RBF}}(\mathbf{x}_i;\mathbf{w},\mathbf{p}) - p(\mathbf{x}_i)\right]^2 \to \min, \quad (6)$$

where $\mathbf{x}_i$ — sampling points (2), λ — matched penalty factor, u_{RBF} — approximate solution obtained at RBFN (3).

The penalty factor λ ensures the fulfillment of boundary conditions, since in meshless methods the conditions at the boundary are not fixed. As can be seen from (6), the use of RBFN allows us to optimize not only the weights, but also the RBF parameters (in the case of the Gauss function, the coordinates of cents and the width). The functional error (6) may include terms with penalty factors that are also responsible for other conditions for the formulation of the problem, for example, relations at media interfaces.

Learning RBFN networks differs from solving the problem of unconditional optimization of the functional (6). Functional (6) is minimized on a limited set of sampling points. A trained network should have the generalization property, that is, pro-

vide a solution with a given accuracy indicator not only at sampling points, but also at arbitrary points in the solution domain. When learning the network, relearning is possible: at sampling points, the accuracy indicator can be small, and at other points it can be large. The possibility of relearning is reduced by using a large number of sampling points. But this approach increases the solution time. The way out is periodic random regeneration of a set of sampling points [14]. From the modern point of view on the learning of neural networks, this technique is the implementation of mini-batch (stochastic) learning [22]. When using sampling point regeneration, the RBFN learning process is organized as a process of minimizing a set of functionals error, each of which is obtained by a specific choice of sampling points. Each functional error is not minimized to the end. Between the regeneration of sampling points, only a few steps are taken of the selected method of minimizing the functional error. This approach circumvents the problem of getting into a local extremum, which is typical for most methods of global nonlinear optimization.

The vast majority of RBFN learning algorithms are based on gradient optimization methods [23]. All gradient methods are local optimization methods, which in general does not guarantee the achievement of a global minimum of the functional error. At the same time, the search for the global minimum of the functional error, generally speaking, is not necessary; it is enough to find the local minimum with some given accuracy. There are known applications of genetic algorithms for learning RBFN networks in solving classification problems [24], which are much simpler than PDE solutions. Three classes are distinguished among gradient methods: zero-order methods that use only the values of the optimized function and not the values of its derivatives during optimization, first-order methods that use the first derivatives of the optimized function (function gradient), and second-order methods that use the second derivatives (Hessian matrix).

Methods to minimize the functional error can be divided into two groups. The first group includes methods for sequentially adjusting weights and RBF parameters. The weights that have the greatest impact on the functionality error are tuned first, then the RBF parameters are tuned. Since the weights enter linearly into the formula for outputting the network (5), optimization methods other than those used for learning RBF parameters that are nonlinear in (5) can be used for their learning.

In the well-known works devoted to solving PDE on RBFN [9–10, 12–14], the simplest first-order method is used — the gradient descent method. Let us consider the implementation of the fastest descent method using the example of the two-dimensional problem (1) and the use of Gaussian as RBF. Consider a single parameter vector RBFN

$$\boldsymbol{\theta} = \left[w_1, w_2, \ldots, w_{n_{RBF}}, c_{11}, c_{21}, \ldots, c_{n_{RBF}1}, c_{12}, c_{22}, \ldots, c_{n_{RBF}2}, a_1, a_2, \ldots, a_{n_{RBF}} \right]^{\mathrm{T}}, \quad (7)$$

where w_j — RBF weights, $j = 1, 2, 3, \ldots, n_{RBF}$, n_{RBF} — number of RBF, c_{j1} and c_{j2} — coordinates of the centers, a_j — width.

Correction of vector (7) at the iteration k in the gradient descent method is carried out according to the formula

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \Delta\boldsymbol{\theta}^{(k+1)}, \quad (8)$$

where $\Delta\boldsymbol{\theta}^{(k+1)} = -\eta\nabla J\left(\boldsymbol{\theta}^{(k)}\right)$ — vector of parameter correction, η — learning speed, selected hyperparameter, $\nabla J\left(\boldsymbol{\theta}^{(k)}\right)$ — the gradient vector of functional (6) over the components of the vector $\boldsymbol{\theta}^{(k)}$ (7) at the iteration k.

Calculations by (8) end with a small value of functional (6). The gradient descent method has a low convergence rate, which does not allow solving problems with high accuracy.

Second-order methods are based on a quadratic approximation of the functional error. In the vicinity of the next approximation of the parameter vector $\boldsymbol{\theta}^{(k)}$ of the network, the functional error (6) is approximated by the Taylor formula

$$J\left(\boldsymbol{\theta}^{(k)} + \Delta\boldsymbol{\theta}^{(k+1)}\right) \approx J\left(\boldsymbol{\theta}^{(k)}\right) + \left[\nabla J\left(\boldsymbol{\theta}^{(k)}\right)\right]^{\mathrm{T}} \Delta\boldsymbol{\theta}^{(k+1)} + \frac{1}{2}\left[\Delta\boldsymbol{\theta}^{(k+1)}\right]^{\mathrm{T}} \mathbf{H}\left(J\left(\boldsymbol{\theta}^{(k)}\right)\right)\Delta\boldsymbol{\theta}^{(k+1)}, \quad (9)$$

where $\nabla J\left(\boldsymbol{\theta}^{(k)}\right)$ — functional gradient, $\mathbf{H}\left(J\left(\boldsymbol{\theta}^{(k)}\right)\right)$ — the Hessian matrix (the matrix of the second derivatives of the functional) calculated with $\boldsymbol{\theta}^{(k)}$.

From the minimum condition for functional (9), the network parameter correction vector $\Delta\boldsymbol{\theta}^{(k+1)}$ can be obtained, which ensures a decrease in the functional error. Due to the complexity of calculating the Hessian matrix for multilayer perceptron, various approximations of the Hessian matrix are used. For example, the conjugate gradient method uses the Fletcher-Reeves formulas (Fletcher R., Reeves C. M.) [25] and Polak-Ribier (Polak E., Ribiere G.) [26]. In quasi-Newtonian methods, the Hessian approximation matrix is calculated at each training step, for example, according to the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula [27]. In the Levenberg-Marquardt method [23], the Hessian matrix is approximated using the product of the Jacobian matrices of the network error vector.

Second-order methods are not widely used in RBFN learning. Although the presence of only one layer with nonlinear functions and the differentiability of most RBFs provide the possibility of applying second-order optimization methods for learning RBFN. In [28], when solving the approximation problem, the nonlinear layer was studied by the conjugate gradient method, and the weights were studied by the method of orthogonal least squares. In [29], an algorithm was proposed for the conjugate gradient adjustment method for RBFN weights, which differs from the known ones taking into account the specifics of solving boundary value problems. RBF parameters were learned by gradient descent method. The algorithm takes into account the differentiability of RBF and is based on the matrix-vector representation of the functional error (6).

In [15], it was proposed, and in [30–31], a fast RBFN learning algorithm was learned, based on an effective optimization method, the trust region method (TRM) [32]. The method allows to simultaneously optimize a large number of parameters, has a high convergence rate even for poorly conditioned tasks, and allows to overcome local minima.

The TRM algorithm is quite complicated, since at least it is found in limited areas, which requires at each step of the optimization process to solve the conditional optimization problem. Therefore, it is advisable to investigate the possibility of adaptation for learning RBFN of modern fast first-order methods and the Levenberg-Marquardt

method. Of particular interest is the Levenberg-Marquardt method, which is simpler to implement than TRM and, as shown in [33], is equivalent to TRM.

3 Development of Levenberg-Marquardt algorithm for learning of radial basis functions networks for solving PDE

The implementation of the Levenberg-Marquardt RBFN learning method for PDE solution will be considered on the example of the model problem described by the Laplace equation with Dirichlet boundary condition

$$\frac{\partial^2 u}{\partial x_1^2}+\frac{\partial^2 u}{\partial x_2^2}=f\left(x_1,x_2\right),\quad \left(x_1,x_2\right)\in\Omega,\quad u=p\left(x_1,x_2\right),\quad \left(x_1,x_2\right)\in\partial\Omega, \tag{10}$$

The functional error for the model problem is the sum of the squared residuals along the internal and boundary sampling points

$$I=\left[\sum_{i=1}^{N}\left(\Delta u_i-f_i\right)^2+\lambda\cdot\sum_{j=1}^{K}\left(u_j-p_j\right)^2\right], \tag{11}$$

where Δu_i — Laplacian at the point i, $r_i=\Delta u_i-f$ —residual of the i-th internal sampling point, $r_j=u_j-p_j$ — residual at the j-th boundary sampling point.

In the Levenberg-Marquardt method, the correction $\Delta\boldsymbol{\theta}^{(k)}$ of the parameter vector $\boldsymbol{\theta}$ (7) is found from the solution of a system of linear algebraic equations

$$\left(\mathbf{J}_{k-1}^{\mathrm{T}}\mathbf{J}_{k-1}+\mu_k\mathbf{E}\right)\Delta\boldsymbol{\theta}^{(k)}=-\mathbf{g}_{k-1}, \tag{12}$$

where $\mathbf{J}_{k-1}^{\mathrm{T}}\mathbf{J}_{k-1}+\mu_k\mathbf{E}$ — an approximation of the Hessian matrix, $\mathbf{E}$ — identity matrix, μ_k — regularization parameter that changes at each step of learning, $\mathbf{g}=\mathbf{J}^{\mathrm{T}}\mathbf{r}$ — gradient vector of functional (11) according to the vector of $\boldsymbol{\theta}$ parameters, $\mathbf{r}=\left[r_1\ \ r_2\ \ldots\ r_n\right]^{\mathrm{T}}$ — residual vector at internal and boundary sampling points, $\mathbf{J}_{k-1}$ — Jacobi matrix calculated in $k-1$ iteration.

Let's represent the Jacobi matrix in block form $\mathbf{J}=\left[\mathbf{J}_{\mathbf{w}}\ \vdots\ \mathbf{J}_{\mathbf{c}_1}\ \vdots\ \mathbf{J}_{\mathbf{c}_2}\ \vdots\ \mathbf{J}_{\mathbf{a}}\right]$, where

$$\mathbf{J}_{\mathbf{w}}=\begin{bmatrix}\frac{\partial r_1}{\partial w_1} & \cdots & \frac{\partial r_1}{\partial w_{n_{RBF}}}\\ \frac{\partial r_2}{\partial w_1} & \cdots & \frac{\partial r_2}{\partial w_{n_{RBF}}}\\ \cdots & \cdots & \cdots\\ \frac{\partial r_n}{\partial w_1} & \cdots & \frac{\partial r_n}{\partial w_{n_{RBF}}}\end{bmatrix},\ \mathbf{J}_{c_1}=\begin{bmatrix}\frac{\partial r_1}{\partial c_{11}} & \cdots & \frac{\partial r_1}{\partial c_{n_{RBF}1}}\\ \frac{\partial r_2}{\partial c_{11}} & \cdots & \frac{\partial r_2}{\partial c_{n_{RBF}1}}\\ \cdots & \cdots & \cdots\\ \frac{\partial r_n}{\partial c_{11}} & \cdots & \frac{\partial r_n}{\partial c_{n_{RBF}1}}\end{bmatrix},$$

$$\mathbf{J}_{c_2}=\begin{bmatrix}\frac{\partial r_1}{\partial c_{12}} & \cdots & \frac{\partial r_1}{\partial c_{n_{RBF}2}}\\ \frac{\partial r_2}{\partial c_{12}} & \cdots & \frac{\partial r_2}{\partial c_{n_{RBF}2}}\\ \cdots & \cdots & \cdots\\ \frac{\partial r_n}{\partial c_{12}} & \cdots & \frac{\partial r_n}{\partial c_{n_{RBF}2}}\end{bmatrix},\ \mathbf{J}_{\mathbf{a}}=\begin{bmatrix}\frac{\partial e_1}{\partial a_1} & \cdots & \frac{\partial e_1}{\partial a_{n_{RBF}}}\\ \frac{\partial e_2}{\partial a_1} & \cdots & \frac{\partial e_2}{\partial a_{n_{RBF}}}\\ \cdots & \cdots & \cdots\\ \frac{\partial e_n}{\partial a_1} & \cdots & \frac{\partial e_n}{\partial a_{n_{RBF}}}\end{bmatrix},$$

where $n=N+K$ — total number of sampling points.

The elements of the Jacobi matrix are easy to calculate analytically. Elements of the $\mathbf{J}_w$ matrix for internal sampling points are calculated by the formula

$$\frac{\partial e_i}{\partial w_j}=\frac{\partial\left(\Delta v_i-f_i\right)}{\partial w_j}=e^{-\frac{\left\|\mathbf{x}-\mathbf{c}_j\right\|^2}{2a_j^2}}\cdot\frac{\left\|\mathbf{x}-\mathbf{c}_j\right\|^2-2a_j^2}{a_j^4}.$$

For boundary sampling points, calculations are performed using the $\frac{\partial e_i}{\partial w_j}=\exp\left(-\frac{\left\|\mathbf{x}-\mathbf{c}_j\right\|^2}{2a_j^2}\right)$ formula. The $\mathbf{J}_{c_1}$ matrix elements for internal sampling points are of the form

$$\frac{\partial e_i}{\partial c_{j1}}=\frac{w_j}{a_j^4}\cdot e^{-\frac{\left\|\mathbf{x}-\mathbf{c}_j\right\|^2}{2a_j^2}}\cdot\left(x_1-c_{j1}\right)\cdot\frac{\left\|\mathbf{x}-\mathbf{c}_j\right\|^2-4a_j^2}{a_j^2}.$$

For boundary points, matrix elements are written as $\frac{\partial e_i}{\partial c_{j1}}=w_j\cdot e^{-\frac{\left\|\mathbf{x}-\mathbf{c}_j\right\|^2}{2a_j^2}}\cdot\frac{(x_1-c_{j1})}{a_j^2}$.

Similarly, the elements of the $\mathbf{J}_{c_2}$ matrix are calculated.

The elements of the $\mathbf{J}_a$ matrix for internal sampling points are of the form

$$\frac{\partial e_i}{\partial a_j}=\frac{w_j}{a_j^5}\cdot e^{-\frac{\left\|\mathbf{x}-\mathbf{c}_j\right\|^2}{2a_j^2}}\cdot\left[\frac{\left\|\mathbf{x}-\mathbf{c}_j\right\|^2}{a_j^2}\cdot\left(\left\|\mathbf{x}-\mathbf{c}_j\right\|^2-2a_j^2\right)-4\cdot\left(\left\|\mathbf{x}-\mathbf{c}_j\right\|^2-a_j^2\right)\right].$$

For boundary points, matrix elements are written as $\frac{\partial e_i}{\partial a_j}=w_j\cdot e^{-\frac{\left\|\mathbf{x}-\mathbf{c}_j\right\|^2}{2a_j^2}}\cdot\frac{\left\|\mathbf{x}-\mathbf{c}_j\right\|^2}{a_j^3}$.

The condition for completing the learning process by the Levenberg-Marquardt method is a small value of the functional error (11).

The matrix $\mathbf{J}_{k-1}^{\mathrm{T}}\mathbf{J}_{k-1}+\mu_k\mathbf{E}$ of system (12) is dense symmetric and positive definite. Therefore, to solve system (12), one can use the Cholesky method [21]. A drawback of the Cholesky method is the use of a lengthy square root extraction operation when performing matrix decomposition. The $\mathrm{LDL}^{\mathrm{T}}$ decomposition method [21] is free from this drawback, which represents the matrix in the form $\mathbf{A}=\mathbf{LDL}^{\mathrm{T}}$, where $\mathbf{L}$ — the lower triangular matrix with the unit main diagonal, $\mathbf{D}$ — the diagonal matrix, and T

— the matrix transpose operation. When decomposing, the square root extraction operation is not applied.

In the Levenberg-Marquardt method, the regularization parameter μ must change during the learning of the network. The learning process begins with a relatively large value of the parameter μ. This means that at the beginning of the learning process, Hessian in (12) is close to the approximate value $\mathbf{H} \approx \mu\mathbf{E}$, and the correction vector is determined by the gradient descent method with a small step $\Delta\boldsymbol{\theta}^{(k)} = -\mathbf{g}_{k-1}/\mu_k$. As the functional error decreases, the parameter μ decreases and the method approaches the Newton method with the Hessian approximation $\mathbf{H} \approx \mathbf{J}^{\mathrm{T}}\mathbf{J}$. This ensures a high convergence rate, since the Newton method near the minimum of the functional error has good convergence. In [33], it is recommended to start with some value of μ_0 and coefficient $\nu > 1$. The current value of μ is divided by ν if the functional error is reduced, or multiplied by ν if the functional error is increased.

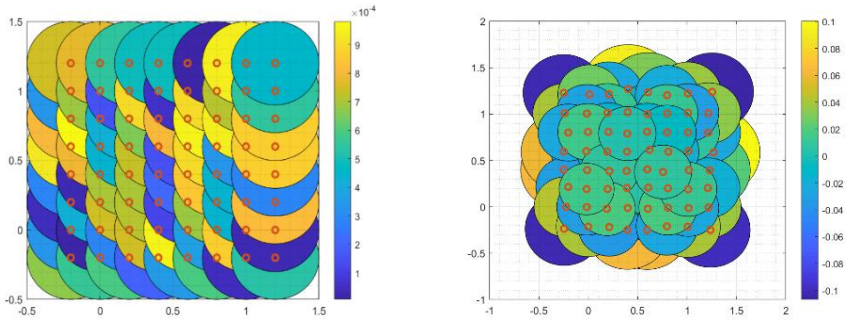
It was shown in [33] that the Levenberg – Marquardt method is equivalent to TRM, and the radius of the trust region is controlled by the parameter μ. But unlike the well-known TRM implementations, the Levenberg-Marquardt method does not require solving a rather complicated conditional optimization problem at each learning iteration. That is, the Levenberg-Marquardt method, while maintaining the positive properties of the trust region method, is simpler.

The disadvantage of the Levenberg-Marquardt method is the poor conditionality of system (12), which depends on the RBF width and increases with increasing accuracy of calculations. It is known [34] that the matrix whose elements are RBF is poorly conditioned and the conditionality of the matrix depends on the width of the RBF. As the RBF width increases, the elements of the matrix $\mathbf{J}_{\mathbf{w}}$ tend to unity, and the elements of the matrices $\mathbf{J}_{\mathbf{c}}$ and $\mathbf{J}_{a}$ tend to zero. The condition number of the matrix $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ is increasing. The regularization parameter μ improves the conditionality of system (12), but a decrease in the parameter μ as the error decreases leads to a deterioration in conditionality.

4 Experiments

An experimental study was carried out using the example of problem (10) with $f(x_1, x_2) = \sin(\pi x_1) \cdot \sin(\pi x_2)$, $p(x_1, x_2) = 0$. The problem was solved in a single square. The number of internal and boundary sampling points is $N = 100$, $K = 40$. The penalty factor is $\lambda = 10$. The RBF centers were regularly located on a square grid with the number of centers at each coordinate equal to 8. Sampling points were randomly located in the solution region and on the region boundary. Weights were initiated by zero values. The initial width of all RBFs was constant, equal to 0.2. The experiments were carried out in the MATLAB R2019a system. To solve system (12), we used the MATLAB system solver. The RBFN learning by the Levenberg-Marquardt method was compared with the gradient descent learning and the accelerated Nesterov method [35] - the fastest first-order method.

In fig. Fig. 1 shows the location of the centers, the symbol of the width (in the form of circles with radii equal to the width) of RBF, and the weights using the MATLAB color palette before learning the network (Fig. 1a) and after learning (Fig. 1b). Fig. 1 shows the importance of setting RBF parameters.



a) before learning the network

b) after learning the network by the Levenberg-Marquardt method

Fig. 1. The centers and width of RB functions in solving PDE

The dependence of the mean square residual of various algorithms on the iteration number is shown in Fig. 2.

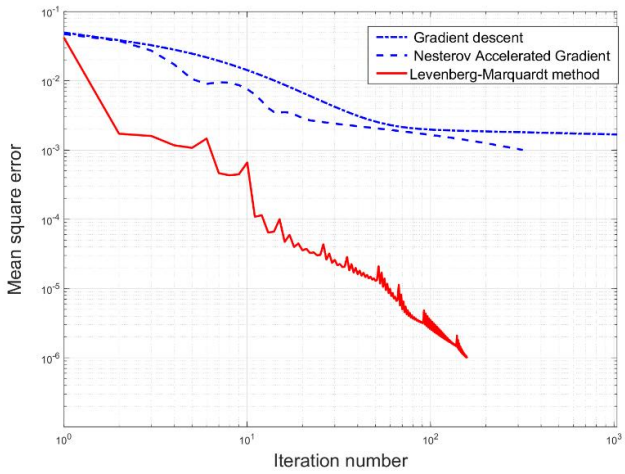


Fig. 2. Dependences of the mean quadratic residual of various algorithms on the iteration number

The results of experiments to solve the boundary value problem on RBFN networks learned by various algorithms are presented in Table 1. The gradient descent method made it possible to solve the model problem with little accuracy. To solve

with high accuracy, the method is practically not applicable. The Nesterov method provides somewhat greater accuracy. Only the Levenberg Marquardt method allowed us to solve the problem with high accuracy in an acceptable time. The Levenberg-Marquardt method showed practically the same results compared to the trust region method [15], but the implementation of the Levenberg-Marquardt method is simpler. The disadvantages of the Levenberg-Marquardt method are the poor conditionality of the system that forms the correction of the parameters, and the non-smooth nature of the convergence.

Thus, the algorithm of the Levenberg-Marquardt method showed a clear advantage over first-order algorithms and ensured accuracy at the level of known implementations of the trust region algorithm, but is simpler than these algorithms.

5 Conclusions

Networks of radial basis functions are a promising means of solving boundary value problems described by partial differential equations. But the well-known methods of learning networks of radial basis functions do not provide quick learning of networks of radial basis functions. As a way to eliminate this drawback, it is proposed to improve the algorithms for learning networks.

For learning networks of radial basis functions intended for solving PDE, a learning algorithm based on the Levenberg-Marquardt method has been developed, which differs by taking into account the specifics of the network architecture and analytical calculation of parameters. The method made it possible to achieve the average quadratic discrepancy, which is not achievable by the known first-order algorithms, on the model problem. The proposed algorithm achieves a small error for the number of iterations equal to the number of iterations of the algorithm based on the trust region method, but is simpler than this algorithm, since it does not require solving the conditional optimization problem at each iteration.

References

1. Grieves, M.: Digital Twin: manufacturing excellence through virtual factory replication. White Paper, 1–7 (2014).
2. Madni, A. M., Madni, Lucero S. D.: Leveraging digital twin technology in model-based systems engineering. Systems 1, Article-Number 7, doi:10.3390/systems7010007 (2019).
3. Farlow, S. J.: Partial differential equations for scientists and engineers. Dover Publications (1993).
4. Mazumder, S.: Numerical methods for partial differential equations: finite difference and finite volume methods. Academic Press (2015).
5. Tolstykh, A. I., Shirobokov, D. A.: Mesh-free method based on radial basis functions. Computational Mathematics and Mathematical Physics 45(8), 1447–1454 (2005).
6. Vasilyev, A., Tarkhov, D., Malykhina, G.: Methods of creating digital twins based on neural network modeling. Modern Information Technologies and IT-Education 14(3), 521–532 (2018).

7. Meshfree methods for partial differential equations. Editors Griebel, M., Marc. A. Schweitzer, M. A. Springer (2008).
8. Buhmann, M. D.: Radial basis functions: theory and implementations. Cambridge University Press (2004).
9. Chen, W., Fu, Z.-J.: Recent advances in radial basis function collocation methods. Springer (2014).
10. Yadav, N., Yadav, A., Kumar, M.: An introduction to neural network methods for differential equations. Springer (2015).
11. Aggarwal, C. C.: Neural networks and deep learning. Springer (2018).
12. Jianyu, L., Siwei, L., Yingjian, Q., Yaping, H.: Numerical solution of elliptic partial differential equation by growing radial basis function neural networks. Neural Networks 16(5–6), 729–734 (2003).
13. Mai-Duy, N., Tran-Cong, T.: Solving high order ordinary differential equations with radial basis function networks. International Journal of Numerical Methods in Engineering, 62, 824–852 (2005).
14. Vasiliev, A. N., Tarkhov, D. A.: Neural network modeling: Principles. Algorithms. Applications, St. Petersburg Polytechnic University Publishing House (2009).
15. Gorbachenko, V. I., Zhukov, M. V.: Solving boundary value problems of mathematical physics using radial basis function networks. Computational Mathematics and Mathematical Physics 57(1), 145–155 (2017).
16. Gorbachenko, V. I., Lazovskaya, T. V., Tarkhov, D. A., Vasiljev A. N., Zhukov, M. V.: Neural network technique in some inverse problems of mathematical physics. Advances in Neural Networks - ISNN 2016: 13th International Symposium on Neural Networks, ISNN 2016, St. Petersburg, Russia, July 6-8, 2016, Proceedings (Lecture Notes in Computer Science). Springer, 310–316 (2016).
17. Fasshauer, G., Zhang, J.: On choosing "optimal" shape parameters for RBF approximation. Numerical Algorithms 45(1–4), 345–368 (2007).
18. Kansa, E. J.: Multiquadrics — A scattered data approximation scheme with applications to computational fluid-dynamics — I surface approximations and partial derivative estimates. Comput. Math. Appl. 19(8–9), 127–145 (1990).
19. Kansa, E. J.: Multiquadrics — A scattered data approximation scheme with applications to computational fluid-dynamics — II solutions to parabolic, hyperbolic and elliptic partial differential equations. Comput. Math. Appl. 19(8–9), 147–161 (1990).
20. Niyogi, P., Girosi, F.: On the relationship between generalization error, hypothesis complexity, and sample complexity for radial basis functions. Neural Computation 8(4), 819–842 (1996).
21. Watkins, D.: Fundamentals of matrix computations. Wiley (2010).
22. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT Press (2016).
23. Gill, P. E., Murray, W., Wright, M. H.: Practical optimization. Emerald Group Publishing (1982).
24. Weikuan, J., Dean, Z., Tian, S., Chunyang, S., Chanli, H., Yuyan, Z.: A New optimized GA-RBF neural network algorithm. Computational Intelligence and Neuroscience. Article ID 982045 (2014).
25. Fletcher, R., Reeves, C. M.: Function minimization by conjugate gradients. Computer Journal 7, 149–154 (1964).
26. Polak, E., Ribiére, G.: Note sur la convergence de méthodes de directions conjuguées. Revue française d'informatique et de recherche opérationnelle, série rouge 3(1), 35–43 (1969).
27. Nocedal, J., Wright, S.: Numerical Optimization. Springer (2006).

28. Zhang, L., Li, K., Wang, W.: An improved conjugate gradient algorithm for radial basis function (RBF) networks modelling. Proceedings of 2012 UKACC International Conference on Control, 19–23.
29. Gorbachenko, V. I., Artyukhina, E. V.: Mesh-free methods and their implementation with radial basis neural networks. Neirokomp'yutory: Razrabotka, Primentnine 11, 4–10 (2010) (in Russian).
30. Alqezweeni, M. M., Gorbachenko, V. I., Zhukov, M. V., Jaafar, M. S.: Efficient solving of boundary value problems using radial basis function networks learned by trust region method. Hindawi. International Journal of Mathematics and Mathematical Sciences. Article ID 9457578 (2018).
31. Elisov, L. N., Gorbachenko, V. I., Zhukov, M. V. Learning radial basis function networks with the trust region method for boundary problems. Automation and Remote Control 79(9), 1621–1629 (2018).
32. Conn, A. R., Gould, N. I. M., Toint, P. L.: Trust-region methods. MPS-SIAM (1987).
33. Marquardt, D.W.: An algorithm for least-squares estimation of nonlinear parameters. Journal of the Society for Industrial and Applied Mathematics 11(2), 431–441 (1963).
34. Boyd, J. P., Gildersleeve, K. W.: Numerical experiments on the condition number of the interpolation matrices for radial basis functions. Applied Numerical Mathematics 61(4), 443–459 (2011).
35. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. ICML'13 Proceedings of the 30th International Conference on International Conference on Machine Learning 28, III-1139-III-1147 (2013).