computation

MDPI

Article

Physics-Based Neural Network Methods for Solving Parameterized Singular Perturbation Problem

Tatiana Lazovskaya [1,2,*] and Galina Malykhina [2,3] and Dmitry Tarkhov [1,2]

[1] Scientific and Technological Centre (STC) "Mathematical Modelling and Intelligent Control Systems", Peter the Great St. Petersburg Polytechnic University, 195251 Saint Petersburg, Russia; dtarkhov@gmail.com
[2] Department of Higher Mathematics, Peter the Great St. Petersburg Polytechnic University, 195251 Saint Petersburg, Russia; g_f_malychina@mail.ru
[3] High School of Cyber-Physical Systems and Control, Peter the Great St. Petersburg Polytechnic University, 195251 Saint Petersburg, Russia
* Correspondence: tatianala@list.ru

Abstract: This work is devoted to the description and comparative study of some methods of mathematical modeling. We consider methods that can be applied for building cyber-physical systems and digital twins. These application areas add to the usual accuracy requirements for a model the need to be adaptable to new data and the small computational complexity allows it to be used in embedded systems. First, we regard the finite element method as one of the "pure" physics-based modeling methods and the general neural network approach as a variant of machine learning modeling with physics-based regularization (or physics-informed neural networks) and their combination. A physics-based network architecture model class has been developed by us on the basis of a modification of classical numerical methods for solving ordinary differential equations. The model problem has a parameter at some values for which the phenomenon of stiffness is observed. We consider a fixed parameter value problem statement and a case when a parameter is one of the input variables. Thus, we obtain a solution for a set of parameter values. The resulting model allows predicting the behavior of an object when its parameters change and identifying its parameters based on observational data.

Keywords: cyber-physical systems; hybrid models; physics-based; physics-informed; stiffness; differential equations; multilayer model; parameterized problem; singular perturbation

Citation: Lazovskaya, T.; Malykhina, G.; Tarkhov, D. Comparing Physics-Based Neural Network Methods for Solving Parameterized Singular Perturbation Problem. *Computation* **2021**, *9*, 97. https://doi.org/10.3390/computation9090097

Academic Editor: Demos T. Tsahalis

Received: 17 July 2021
Accepted: 1 September 2021
Published: 6 September 2021

1. Introduction

The development of technologies that make it possible to obtain data from environments, observed objects, and processes and process them with sufficient speed and efficiency leads to the widespread implementation of cyber-physical systems (CPSs) [1]. As a part of the operating of these systems, the received information (data) is synchronized with a corresponding physical or information object. The result of such synchronization is a digital twin (DT) of an object, i.e., a model of a complex system. The DT continues to synchronize with the modeled object and reflect its dynamic characteristics that change over time. Such models can be used to conduct digital experiments, implement alternative external scenarios, and predict the behavior of an object in the future without affecting or changing a real prototype.

The issue of building DTs and the simultaneous development of artificial intelligence technologies have led to the emergence of so-called "pure" data-driven modeling "black boxes", in which a model is created on the basis of deep learning using big data. While the possibility of obtaining such data has been growing in recent times, the cost of storing and processing data has increased along with a rise in data volumes. In addition, for some complex industrial and technological processes, the acquisition of a large amount of data itself can be accompanied by high expenditure and, sometimes, a lack of measurement

possibilities. In these cases, the data can be utilized to verify a model built by classical modeling methods based on the physics of an object, including numerical methods for solving corresponding problems of mathematical physics.

The intention to preserve the advantages and eliminate the disadvantages of both types of methods is reflected in the new classifications of existing modeling paradigms [2–4]. Physical laws provide interpretability and extrapolation properties to data-driven models which, in turn, connect theoretical models to real objects. We do not aim at repeating or improving the overviews of different models. However, the description of various modeling classes seems essential. Ref. [2] reviews the concepts of DT modeling and highlights the value of the DT in the modern world and its application in various sectors of the economy. Among the models combining physics and data-based modeling, [2] points out four categories formed by the mutual integration of the following modeling approaches: physics-based modeling, data-driven modeling, and big data methods. A similar classification can be found in [5]. The article [3] focuses on the CPS application and provides a detailed classification of hybrid models, distinguishing among physics-based machine learning modeling classes of algorithms that differ in the nature of interaction between physics-based and machine learning (ML) approaches to modeling. Firstly, these are the methods that utilize physics theory for the preprocessing of data, which then arrive at an input of a neural network in ML. The second type of model comprises those in which prior knowledge of the physics of an object is reflected in an architecture of a network. Thirdly, one of the most popular hybrid modeling methods is underlined, which consists of using physics equations as an additional regularization term in the loss function of an ML model. Similar classes are described in [6]. In [3], models in which activation functions are physically interpretable also act as a separate class. In [4,7], in addition to those mentioned above, embedding approaches are pointed out, in which data-driven models are used as components of classical physics-based models to speed up, simplify, or replace a numerical solution.

The authors of one of the most cited recent papers on hybrid modeling of complex systems [8] mention that, among others, they have been inspired by the work [9] dedicated to the use of neural networks to solve ordinary and partial differential equations. Let us note that in our work [10], published in 2005, we talk about solving problems of mathematical physics by using both neural networks and real measurements. From the point of view of modern classification, we can regard this approach as a hybrid model with physics-based regularization.

In this article, we consider our previously developed methods of hybrid modeling and their modifications in the context of the modern classification of paradigms for modeling complex dynamic systems. We present the principles of building physics-based network models naturally capable of further adaptation and refinement using measurement data and other additional data.

We have chosen the problem of modeling a system described by a boundary value problem with an unknown parameter as a specific issue of this paper. Problems involving differential equations with unknown parameters frequently arise in the modeling and control of CPS. These parameters can be regarded as dynamic and changing with time according to an unknown law. When describing complex systems, the parameter can characterize the conditions in which the object exists, or a specific part of this object. Thus, the parameter can influence a type of the differential problem under question. It can be considered a hard problem, an ill-posed problem, a problem with a small parameter at the highest derivative, and, accordingly, a problem with a boundary layer, and other types. Let us note that, when solving a problem with an unknown parameter, we do not set to ourselves the task of identifying this parameter, dubbed data-driven discovery of differential equations. As a result of applying methods described below, we obtain a parametric solution that is universal for various parameter values belonging to a predetermined interval that is used in the construction and training of a neural network solution. The resulting pre-trained model can be further adapted by training on the measured data.

Based on these measurements, unknown parameters can also be estimated by any means or adjusted using additional training of the network. A similar problem is solved in [11], where simulating a lake's temperature is considered. The authors of [11] use the loss function with physics-based regularization to pre-train a recurrent neural network model before adapting it by means of measurements of a certain lake. They also suggest using the finite element method data as a training dataset.

It is necessary to make a remark about the accuracy of a solution to the problem. Nowadays, there are two fundamentally different approaches to mathematical modeling. The first one is traditional and involves passing from a simulated object to a model in the form of a differential equation (system of equations) with additional conditions (initial, boundary, etc.). Further, this model is considered the object itself and an approximate solution is constructed based on it. The higher the accuracy of such a solution, the more accurate the model of the original real object appears to be. The second approach, which we adhere to, is that the differential model built on the basis of physical principles is certainly approximate. For real complex technical objects, the accuracy of similar models is often low. Therefore, the solution of differential equation(s) with high accuracy has no practical sense. More promising is the construction of an adaptive model that corresponds to the differential equation and additional conditions with reasonable accuracy and the subsequent adaptation of this model to the data of object observations. This approach is especially relevant in a situation when the properties of an object can change during its operation.

The paper [12] considers hybrid systems, which are a class of problems that cyclically switch between two phases: the first is described by a stiff system of differential algebraic equations of motion (interaction with the ground), the second corresponds to the flight phase and is specified by a non-stiff system. Using a specific problem (hopping single-legged robots) as an example, the authors of [12] compare various ODE (ordinary differential equation) solvers, including those common in the Matlab package. It is noted that although different solvers work well for different types of problems, due to the constant restart of the numerical solver, the accuracy of the solution diminishes and the cost of computation increases. This problem could be solved with a simple universal approach. Ref. [13] considers the need to solve a parameterized problem that also arises when studying a wide range of complex physical phenomena, when, while modeling many different possible implementations of the system, large requirements for computational resources lead to the forced use of the apparatus for reducing parametric models. The authors underline the importance of such parametric models in global problems of design, control, optimization, and quantification of uncertainty. The development of a controller that works effectively for all values of model parameters is regarded as an urgent task. Ref. [14] proposes a solution to a parameterized differential problem by the FEM for an interval parameter based on interval arithmetic. However, as a result they obtain only the corresponding interval boundaries of the solution. Earlier, the authors of the given article studied the construction of a unified solution to one parameterized differential problem, which, for some values of the parameter, is stiff [15]. The unified neural network approach was applied using various additional information of heterogeneous types to improve the quality of an approximate solution. In [16], we successfully applied this approach to solve various types of problems with parameters. A stiff differential problem and a differential algebraic problem with different numbers of solutions were solved. In the catalyst problem, we solved the problem in a region that is wider than the domain of existence of the solution to the differential equation. The authors of [17] also used trainable neural networks with a single hidden layer to obtain a differentiable closed-form solution for a class of parameterized initial-value systems with varying degrees of stiffness. Ref. [18] highlights the growing interest in the development of numerical methods for solving similar problems and suggests a method based on a rational spectral collocation. Adaptive grid methods are considered in [19–22]. Ref. [23] proposes a special neural network for solving linear singularly perturbed differential equations with initial and boundary values.

We note the advantage of low computational costs of the application of neural network models after the training phase. However, when describing systems with uncertainty, the structure of the differential problem itself may often be unknown in advance. Thus, the need arises for building an analytical solution that does not require real-time training and reflects the model under consideration for different parameter values. This approach related to physics-based network architectures is presented in this paper as an extension of the multilayered techniques [24].

We have planned the structure of this manuscript as follows. Section 2 discusses the methods of constructing the models of systems described in the form of a classical non-parameterized boundary value problem. Through expansion in basis functions, the basic principles of the finite element method (FEM) [25] as one of the "pure" physics-based modeling methods, and the general neural network approach [10] as a variant of ML modeling with physics-based regularization, are presented. In Section 2.3, the ML model is proposed to be improved by using prior knowledge of the physics of an object in the form of a training data set. The values of an FEM solution in the nodes are utilized instead of measurement results for additional training of the neural network model. A less popular modeling class, named physics-based network architectures, is suggested in Section 2.4 as a special multilayered model based on classical numerical methods. Section 3 is devoted to methods of constructing models in cases where the parameters included in the corresponding equations may not be known. In particular, guidance for using the methods described in Sections 2.3 and 2.4 for parameterized problem setting is discussed. We want to emphasize that this certain problem statement is our major interest in this work. Approaches to solving the problem with a fixed parameter are presented to facilitate an understanding of methods for solving parameterized problems. Section 4 features the parameterized singular perturbation problem and presents the results of applying the previously described methods to the model construction. Section 5 contains conclusions and considerations for future research.

2. Materials and Methods. Ordinary Problems

To clarify the methods, let us formulate as an example a boundary value problem

$$\begin{cases} Au = g, \ u = u(\mathbf{x}), \ \mathbf{x} \in \Omega \subset \mathbb{R}^p; \\ Bu|_\Gamma = f; \end{cases} \tag{1}$$

where A is a differential operator, i.e., an algebraic expression containing derivatives of an unknown function u, B is some operator specifying the boundary conditions, and Γ is the boundary of some domain.

We seek an approximate solution in the form of an expansion in basis functions v_i

$$u_N(\mathbf{x}) = \sum_{i=1}^{N} c_i v_i(\mathbf{x}). \tag{2}$$

Searching for a solution in form (2) is usually called the Galerkin method. Its special cases include different variants of the FEM and neural network models, in which equations based on physics are used as an additional term for the regularization of the loss function. In this case, this models differ in the features of the selection of basis functions and parameters c_i.

2.1. Methods with Fixed Basis Functions

First, let us distinguish a group of methods in which basis functions $v_i(\mathbf{x})$ are fixed and only parameters c_i are adjusted. The extreme learning machine (ELM) algorithm [26] for single hidden layer feedforward neural networks (SFLNs) randomly chooses internal weights and biases of basis functions and then analytically obtains the output weights

c_i. Ref. [27] proposes the incremental method for SFLNs and also shows its efficiency for SLFNs with piecewise continuous basis functions.

A similar approach is typical for the simplest version of the FEM. There are two conventional approaches to finding the parameters c_i. The first, the most common of them, is that the space with the dot product $a(\mathbf{x}) \cdot b(\mathbf{x})$ is regarded as the space in which the solution is sought. The simplest version of such a dot product is the product in the integral form $a(\mathbf{x}) \cdot b(\mathbf{x}) = \int_\Omega a(\mathbf{x}) b(\mathbf{x}) d\mathbf{x}$. In this way, the parameters c_i are found from the condition of orthogonality of the residual of Equation (1) to all the basis functions

$$(Au_N(\mathbf{x}) - g(\mathbf{x})) \cdot v_i(\mathbf{x}) = 0. \tag{3}$$

It is easy to ascertain that they can be found as a solution to the system

$$\sum_{i=1}^{N} c_i Av_i(\mathbf{x}) \cdot v_j(\mathbf{x}) = g(\mathbf{x}) \cdot v_j(\mathbf{x}). \tag{4}$$

If the integral cannot be calculated explicitly, it can be calculated by using cubature formulas or through the mean value

$$a(\mathbf{x}) \cdot b(\mathbf{x}) \cong \frac{1}{M} \sum_{j=1}^{M} a(\xi_j) b(\xi_j). \tag{5}$$

Basis functions usually satisfy the homogeneous boundary condition $Bv_i|_\Gamma = 0$. To satisfy the inhomogeneous boundary condition, additional term $v_0(\mathbf{x})$ is added to sum (2). This term satisfies the inhomogeneous boundary condition $Bv_0|_\Gamma = f$. The sum takes the form $u_N(\mathbf{x}) = v_0(\mathbf{x}) + \sum_{i=1}^{N} c_i v_i(\mathbf{x})$.

Accordingly, the substitution $u_N(\mathbf{x})$ into (3) yields a linear system of the form

$$\sum_{i=1}^{N} c_i Av_i(\mathbf{x}) \cdot v_j(\mathbf{x}) = (g(\mathbf{x}) - Av_0(\mathbf{x})) \cdot v_j(\mathbf{x}),\ j = 1, \ldots, M. \tag{6}$$

Or, for the discrete form of the loss function,

$$\sum_{i=1}^{N} c_i \sum_{k=1}^{M} Av_i(\xi_k) \cdot v_j(\xi_k) = \sum_{k=1}^{M} (g(\xi_k) - Av_0(\xi_k)) \cdot v_j(\xi_k),\ j = 1, \ldots, M. \tag{7}$$

This approach is referred to as the collocation method.

The above basis functions are usually chosen so that their supports intersect only for a small number of functions, while the matrix of system (4) is sparse, which greatly speeds up the solution procedure.

The second approach is that the quality of the solution to problem (1) is characterized by a loss function (dubbed the error functional or error function) J, the minimization of which leads to the needed parameters c_i.

As shown above, we can use the integral form of the loss function, for example, $(Au_N(\mathbf{x}) - g(\mathbf{x})) \cdot (Au_N(\mathbf{x}) - g(\mathbf{x}))$.

If it is difficult to calculate the dot product explicitly, then we can use the discrete form

$$J = \sum_{j=1}^{M} (Au_N(\xi_j) - g(\xi_j))^2. \tag{8}$$

The finite element method is characterized by using basis functions (elements) with a compact support localized in small subdomains of Ω . In this case, the supports of the basis functions intersect and cover the entire domain Ω. With this method, if the supports of the basis functions intersect only with a small number of supports of other basis functions, the matrix of the system turns out to be sparse.

For piece-wise linear elements, direct application of Formula (4) may be impossible due to the absence of derivatives of the required order. In several cases, this problem can be solved by applying the Stokes formula or by the corresponding version of the integration by parts formula.

2.2. Physics-Based Neural Network Approach

When solving stiff problems, the previously considered approach with fixed basis functions does not work well, and these functions have to be adjusted during the solving process. In [28], the ELM learning algorithm with a tunable basis function is studied.

At first glance, the general neural network approach is similar to the previous ones, only expansion (2) is specified by the expression

$$u_N(\mathbf{x}) = \sum_{i=1}^{N} c_i v(\mathbf{x}, \mathbf{a}_i). \tag{9}$$

By solving the optimization problem for a certain loss function, both linear parameters c_i and vector parameters (weights and biases) $\mathbf{a}_i$ are adjusted. Functions typical of neural networks are used as basis functions. These are usually radial basis functions (RBFs) or perceptron-type functions with sigmoid basis functions.

2.2.1. Multilayer Perceptron

This type of neural network is the most widely used and researched. Let us describe its structure. Let the input be an m-dimensional vector. Linear combinations of coordinates of the input of the form (w, x) are fed to the input of the first layer of neurons. The coefficients of these combinations are named the weights of the first layer. Each neuron acts as a one-dimensional non-linear function dubbed an activation function. Linear combinations of outputs of neurons are fed to the next layer, and linear combinations of outputs of neurons of the last layer form the output of the network. An additional neuron, whose output always equals 1, is often added to all or some of the layers. Its meaning is the same as adding bias to linear regression, i.e., in fact, the subtraction of a constant value. Experience has shown that this greatly improves network training. In addition, an input is often added, whose value always equals 1. Usually, an activation (basis) function is a function that approximates in form to $\operatorname{sign}(x)$. Its meaning is to split the space into two half-spaces by hyperplanes $(\mathbf{w}, \mathbf{x}) = 0$ and $(\mathbf{w}, \mathbf{x}) + w_0 = 0$ as long as a bias is used. In one of the half-spaces, the output of the neuron is equal to +1; in the other it is equal to -1. If we take not only one neuron, but a layer of neurons, the entire space is divided by hyperplanes into subsets, each of which corresponds to its own set of outputs of neurons of this layer. In this way, the network output is obtained as a piecewise constant function. Usually, instead of function $\operatorname{sign}(x)$, smooth functions are used, which makes it possible to calculate the derivatives and apply gradient methods to train the network. In this case, it is advisable that the calculation of the derivatives of the activation function requires a minimum of additional operations. For such functions, the network does not produce a piecewise constant, but a smoothed dependence. Here are some examples.

The activation function recommended in the works of A.N. Gorban and his students has the form $\varphi(x) = \frac{x}{1+|x|}$ and $\varphi'(x) = \frac{1}{(1+|x|)^2} = (1 - |\varphi(x)|)^2$. One function is more commonly used, namely $\varphi(x) = \operatorname{th}(x)$. Here, $\varphi'(x) = \frac{1}{\operatorname{ch}^2(x)} = 1 - \varphi^2(x)$. An asymmetric version of this function is sometimes used $\varphi(x) = \frac{\exp x}{1+\exp x}$, where $\varphi'(x) = \frac{\exp x}{(1+\exp x)^2} = \varphi(x)(1 - \varphi(x))$. Other functions are also applied, for example, $\varphi(x) = \arctan(x)$, $\varphi'(x) = \frac{1}{1+x^2}$.

Let us describe the functioning of the multilayer perceptron more formally. Denote the vector of inputs of the l-th layer $\mathbf{y}_{l-1}$, and the vector of outputs $\mathbf{x}_l$. Then the network is described by recurrence relations $\mathbf{y}_l = \mathbf{W}_l \mathbf{x}_l$, $\mathbf{x}_l = \varphi_l(\mathbf{y}_{l-1})$. Here $\mathbf{x}_0 = \mathbf{x}$ is a network input and $\mathbf{y}_L = \mathbf{f}(\mathbf{x})$ is a network output. $\mathbf{W}_l$ is a matrix of weights of the l-th layer, and φ_l is the activation function, which acts coordinate-wise.

Application of gradient methods of network training, i.e., minimizing the error, requires calculating the derivatives of the network output with respect to the weights, which are easy to obtain using the formula for differentiating a complex function.

To determine the desired derivatives, calculations are performed using the formula

$$\frac{\partial \mathbf{f}}{\partial w_{ij}^{(l)}} = \mathbf{W}_L \mathbf{Z}_L \mathbf{W}_{L-1} \mathbf{Z}_{L-1} \cdots \mathbf{W}_{l+1} \frac{\partial \mathbf{W}_l}{\partial w_{ij}^{(l)}} \mathbf{x}. \tag{10}$$

Matrices $\mathbf{Z}_L$ are calculated by counting the network outputs. In accordance with (10), computations begin at the last layer and move to the first. In the literature on neural networks, the calculation of derivatives by this algorithm is often combined with calculations by the gradient descent method, and this hybrid is called the back-propagation method. This algorithm is generalized in the form of automatic differentiation and is widely used in training neural networks, including in [8].

2.2.2. Radial Basis Function Networks

The main difference between this type of network and the perceptron is that each neuron is responsible for local approximation in the vicinity of some point, and not for the difference between the values of the function in half-spaces, as in the case of the perceptron. Hence the conclusion that the perceptron is best used to simulate problems with jumps, such as phase transitions and shock waves. RBF networks are preferred for applications where a sufficiently smooth solution is to be expected.

In this case, the activation function has the form $v_i = \varphi(\alpha_i ||\mathbf{x} - \mathbf{x}_i||)$. Here, $\mathbf{x}_i$ is the center weight, α_i is a scale parameter for each neuron of a hidden layer, and $||*||$ is some metric on $\mathbb{R}$. Parameters c_i, α_i, and $\mathbf{x}_i$ are selected during network. Two fundamentally different approaches can be applied to the adjustment of centers.

We consider fixed centers in the distribution domain of the input. They can be chosen regularly in this area according to certain principles or randomly due to some probabilistic law. Each $\mathbf{x}_i$ acts the same as parameters c_i and α_i included in the process of loss function optimization.

There are also two options for choosing parameters α_i. First, $\alpha_i = \alpha$, where α is a certain size of the considered domain. This method is utilized, as a rule, if the centers are selected in a regular way. Alternatively, α_i is defined by some method of non-linear optimization in conjunction with other parameters. We can apply an intermediate option assuming that $\alpha_i = \alpha$ and adjusting α as one of the optimized parameters. There are a few options left for c_i. If they are found separately, then a linear system is obtained. If we mutually select all the coefficients, then we can use some non-linear optimization algorithm.

To solve problem (1) it is natural to choose the loss function in the form $J = J_1 + \delta J_2$, where J_1 corresponds to obeying the equation and J_2 to obeying the boundary condition. Usually, due to the impossibility of calculating integrals explicitly, a discrete form is used. Thus, $J_1 = \sum_{j=1}^{M} (Au_N(\xi_j) - g(\xi_j))^2$ and $J_2 = \sum_{k=1}^{K} (Bu_N(\xi'_k) - f(\xi'_k))^2$.

Our experience of applying neural networks [29] allows making some remarks about the choice of points $\{\xi_j\}_{j=1}^{M}$ and $\{\xi'_k\}_{k=1}^{K}$. They can be chosen inside Ω (or in a wider set $\tilde{\Omega} \supset \Omega$ if it is necessary to ensure sufficient smoothness of an approximate solution up to the boundary) and on the boundary in a regular way; for example, uniformly in the case of a bounded domain or according to the normal probability law if it is unbounded.

It is often more appropriate to utilize a set of points selected according to the probabilistic distribution law and re-generated after a certain number of learning epochs (optimization steps) using a constant (or other) probability density. This ensures more stable learning.

We had previously shown (see, for example, [15]) that the optimal number of steps in the learning process between point regeneration is from 3 to 5. In addition, this approach allows us to control the quality of training using standard statistical procedures. In this way, the number of sample points can be significantly reduced without compromising accuracy [29]. At the same time, the learning process becomes more stable and does not get

stuck in local minima. In fact, not one function but a sequence of functions is optimized, and each is a discrete approximation to the integral form of the loss function.

If the model must satisfy additional conditions, for example, measurement data $u(\xi_l'') = u_l$, a term $\tilde{\delta}J_3$ is added to the loss function. Thus, $J = J_1 + \delta J_2 + \tilde{\delta}J_3$, where $J_3 = \sum_{l=1}^{L}\left(u(\xi_l'') - u_l\right)$.

We have dedicated a number of works to the study of modeling problems using physics-based neural networks, see [10–30]. A detailed description of the methods can be found in [29]. In recent years, physics-based ML modeling methods have included different types of neural networks. Ref. [11] considers recurrent neural networks and applies physics-based regularization. Ref. [31] solves the problem of modeling non-linear dynamical systems described by partial differential equations without initially given training sets. They utilize data generated on the basis of equations to train auto-regressive networks.

Note that the FEM can be regarded as a special case of the RBF network method (ELM). Most often, when using the FEM, piece-wise linear approximation is applied. In this case, triangulation is performed; that is, the domain is divided into triangles (the boundary is approximated by a broken line) and a linear function is constructed on each triangle. On the boundary of the triangles, the functions are continually joined. To obtain a representation of the solution in the form of an expansion in basis functions (2), one can consider the behavior of the function along the rays emanating from a certain center c_i. In the two-dimensional case, we have $v_i = v_i(\rho, \psi)$, where (ρ, ψ) are the polar coordinates of the vector $\mathbf{x} - \mathbf{c}_i$, and $\mathbf{x}(\rho, \psi)$ is the current point. If the position of the boundary of the support of the basis function relative to its center is characterized by some function $\rho = a_i(\psi)$, we can utilize $\varphi_i = (1 - \rho/a_i(\psi))_+$, where $w_+ = w(\mathrm{sign}(w) + 1)/2$. In this case, for the polygonal boundary, we obtain a piecewise linear function, for which $a_i(\psi)$ is calculated from the polar equations of the corresponding straight lines, i.e., at the corresponding intervals of the variation of ψ we obtain $a_i(\psi) = \rho_i / \cos(\psi - \psi_i)$. If we want the basis function on the boundary to have a zero derivative, we can use $\varphi_i = \left((1 - \rho/a_i(\psi))_+\right)^2$. For a basis function of the form $\varphi_i = (1 + 2\rho/a_i(\psi))\left((1 - \rho/a_i(\psi))_+\right)^2$, we obtain a smooth vertex. Some more complicated form of a basis function allows obtaining a smooth surface with a polygonal base, but we will not go into that.

In the multidimensional case, we can use $v_i = v_i(\rho, \mathbf{v})$, where $\rho = \mathbf{x} - \mathbf{c}_i$ and $\mathbf{v}$ is a vector on the unit sphere, which can be parameterized by the corresponding coordinates.

2.3. Physics-Based Neural Network Approach with FEM Smoothing

Both physics-based models considered, the FEM and ML with physics-based regularization, have their advantages and disadvantages. Higher degrees of polynomial elements lead to an increase in the cost of computation, although we obtain a solution with higher smoothness. In any case, it is not possible to obtain a solution of infinite smoothness as for neural networks.The FEM is long and more extensively applied. It is implemented in numerous commercial software products. However, the FEM using piecewise linear functions leads to an insufficiently smooth solution. In addition, if the number of elements of the method is small, the accuracy is not high enough; if it is large, the system of linear equations may require large computational resources even with a sparse matrix of system (4).

The FEM is poorly suited to solving problems in which additional data are used to build a model. This is especially the case for tasks in which the model must adapt to the newly received data, taking into account changes in the modeling object. This is largely due to the local nature of the basis functions, which leads, in turn, to a local restructuring of the solution when trying to adapt it by minimizing the loss function J_3. The neural networks, by their nature, are capable of being used in the problem of adapting a solution to new data. This is evidenced by the extensive utilizing of neural networks to extract the information hidden in noisy data. However, the direct application of the above neural network approach often leads to a computationally complex problem of non-linear optimization. If only parameters c_i of a neural network solution are adjusted, a system

of linear equations with a dense matrix is obtained, which dramatically increases the complexity of its solution. A more efficient approach is to first optimize the loss function $J = J_1 + \delta J_2$, and then train the neural network when new data arrive by minimizing J_3. However, this method is also very resource-intensive.

Alternatively, we propose the following hybrid approach. First, a solution to problem (1) is constructed using the FEM. Further, this solution is smoothed by means of a neural network. For the approach discussed above, all the basis functions except one are equal to zero at each nodal point, and this type of model is the easiest to build. It consists of optimizing J_3 in which the values of the FEM solution at the nodes are used instead of the measurement results. The neural network approximate solution obtained can be adapted to the newly received data in the same way as described above. Let us emphasize that in this hybrid model several physics-based modeling classes are combined.

In [32], a FEM solution is also used as training data in the problem of structural damage identification. However, in that work, the values at nodes are not used directly, but the loss function includes two terms with the corresponding weight factors. The first is responsible for a cross-entropy loss and the second corresponds to normalized damage probabilities.

2.4. Physics-Based Multilayer Network Architecture Method

The expressions of approximate solutions constructed by using the above methods are rather cumbersome. If we write them out analytically, they turn out to be poorly visible. Much more compact approximations are obtained through an approximate series expansion or various kinds of asymptotic methods [33]. However, these methods make it possible to build an acceptable solution only in a sufficiently small neighborhood of the point at which the expansion is constructed or for sufficiently small values of the parameter over which the expansion is carried out. In [24] we propose another approach, which has an essentially broader area of applicability. From the point of view of the modern modeling classification, we can attribute this method to physics-based network architectures. The resulting analytical models are fairly compact.

Let us consider the initial value problem

$$\begin{cases} y^{(n)}(x) = f(x, y(x), y'(x), \dots, y^{(n-1)}(x)), \\ y(x_0) = y_0, \\ y'(x_0) = y'_0, \\ \dots \\ y^{(n-1)}(x_0) = y_0^{(n-1)}, \end{cases} \qquad x \in [x_0, x_0 + a]. \tag{11}$$

Using the replacement, one can go from a higher-order system to a first-order multi-dimensional problem

$$\begin{cases} \mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x)), \\ \mathbf{y}(x_0) = \mathbf{y}_0, \end{cases} \quad x \in [x_0, x_0 + a]. \tag{12}$$

To solve (12), many numerical methods have been developed [34]. A significant part of them consists of the partition of a given interval by points x_k into intervals of length h_k, $k = 1, \dots, n$, and applying the recurrence formula

$$\mathbf{y}_{k+1} = \mathbf{y}_k + F(\mathbf{f}, h_k, x_k, \mathbf{y}_k, \mathbf{y}_{k+1}), \tag{13}$$

where operator F defines a specific method. If in (13) the function F depends on $\mathbf{y}_{k+1}$, it is necessary to accurately or approximately solve (13) with respect to $\mathbf{y}_{k+1}$.

Applying formula (13) iteratively n times to an interval with a variable right end $[x_0, x] \subset [x_0, x_0 + a]$ we obtain function $\mathbf{y}(x)$, which can be considered an approximate solution of system (12). Let us underline that, in this case, $h_k = h_k(x)$, $\mathbf{y}_0 = \mathbf{y}_0(x)$, $\mathbf{y}_k = \mathbf{y}_k(x)$.

For constructing multilayer solutions, we can choose one of the classical numerical methods [34], for example:

- the implicit Euler method, where

$$F(\mathbf{f}, h_k, x_k, \mathbf{y}_k, \mathbf{y}_{k+1}) = h_k \mathbf{f}(x_{k+1}, \mathbf{y}_{k+1}); \tag{14}$$

- the trapezium method, where

$$F(\mathbf{f}, h_k, x_k, \mathbf{y}_k, \mathbf{y}_{k+1}) = \frac{1}{2} h_k (\mathbf{f}(x_k, \mathbf{y}_k) + \mathbf{f}(x_{k+1}, \mathbf{y}_{k+1})); \tag{15}$$

- the midpoint method, where

$$\begin{cases} \mathbf{y}_{k+2} = F(\mathbf{f}, h, x_k, \mathbf{y}_k, \mathbf{y}_{k+1}), \\ F(\mathbf{f}, h, x_{k+1}, \mathbf{y}_k, \mathbf{y}_{k+1}) = \mathbf{y}_k + 2h\mathbf{f}(x_{k+1}, \mathbf{y}_{k+1}). \end{cases} \tag{16}$$

or one of many other methods of this kind.

If function $\mathbf{f}(x, \mathbf{y})$ is a neural network (for example, it was obtained by modeling some related process), the result is an arbitrarily accurate solution to the original problem. In this case, we refer the entire modeling procedure to the hybrid methods mentioned in the Introduction as the embedding approach. Otherwise, it can be approximated by a neural network (for sufficiently wide classes of functions, this can be done with arbitrarily high accuracy) and the equation can be solved with the replacement of the right-hand side by its approximation.

The resulting models can be regarded as physics-based pre-trained multilayer networks, the numerical parameters of which can be further trained using measurement data.

Let us consider the boundary value problem

$$\begin{cases} \mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x)), \\ \mathbf{u}(x_0) = \mathbf{u}_0, \ \mathbf{v}(x_0 + a) = \mathbf{v}_0, \end{cases} x \in [x_0, x_0 + a]. \tag{17}$$

Here, vectors $\mathbf{v}$ and $\mathbf{u}$ are composed of the coordinates of vector $\mathbf{y}$; their total dimension is equal to the dimension of $\mathbf{y}$.

The following approach turned out to be an effective way to solve this problem. We choose arbitrary point $t \in (x_0, x_0 + a)$ and build a solution to the system

$$\begin{cases} \mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x)), \\ \mathbf{y}(t) = \mu, \end{cases} x \in [x_0, x_0 + a]. \tag{18}$$

where parameter μ is adjusted based on the boundary conditions

$$\begin{cases} \mathbf{u}(x + 0) = \mathbf{u}_0, \\ \mathbf{v}(x_0 + a) = \mathbf{v}_0. \end{cases} \tag{19}$$

Parameter t remains undefined. It can be chosen so that a solution satisfies the original system with the highest accuracy. For the problems of modeling real objects and constructing DT, it is more natural to adjust t based on the best fit to the measurement data.

These multilayer models can easily adapt to new data about an object. For this purpose, we can change t and μ or only μ and minimize the loss function $\sum_{l=1}^{L} ||\mathbf{y}_n(\xi_l'') - \mathbf{y}_l||^2$. $||*||$ takes into account only those coordinates of $\mathbf{y}_l$ for which measurement data are available. We have also studied the adaptive properties of the multilayer model in [35].

In [36], another scheme for imposing physics constraints on the neural network architecture is applied. They introduce a method that converts different types of equations into linear constraints and utilize them to build the output layer of a neural network. For inequality conditions, [36] suggests using positive definite activation functions.

3. Materials and Methods. Parameterized Problems

The formalization of parameterized problems is that some parameter $\mathbf{r}$ appears in the formulation of problem (1) in that $A = A(\mathbf{r})$, $B = B(\mathbf{r})$, $\Gamma = \Gamma(\mathbf{r})$, $\Omega = \Omega(\mathbf{r})$. It is required to find a solution $u(\mathbf{r})$ for which

$$\begin{cases} A(\mathbf{r})u = g,\ u = u(\mathbf{x}),\ \mathbf{x} \in \Omega(\mathbf{r}) \subset \mathbb{R}^p; \\ B(\mathbf{r})u|_\Gamma(\mathbf{r}) = f; \end{cases} \tag{20}$$

holds for all $\mathbf{r}$ from a given set.

3.1. FEM and Physics-Based Neural Network Approach

Direct application of the FEM to solving problem (20) leads to replacing (2) by an expansion

$$u_N(\mathbf{x}, \mathbf{r}) = \sum_{i=1}^{N} c_i(\mathbf{r}) v_i(\mathbf{x}, \mathbf{r}), \tag{21}$$

and system (4) is replaced by system

$$\sum_{i=1}^{N} c_i(\mathbf{r}) A(\mathbf{r}) v_i(\mathbf{x}, \mathbf{r}) \cdot v_j(\mathbf{x}, \mathbf{r}) = (g(\mathbf{x}, \mathbf{r}) - A(\mathbf{r}) v_0(\mathbf{x}, \mathbf{r})) \cdot v_j(\mathbf{x}, \mathbf{r}). \tag{22}$$

As a result, we have a system of linear equations with coefficients depending on the parameter. There are three possibilities among physics-based methods to solve it. The first one is analytical. The second approach uses interval analysis. These techniques work in a reasonable amount of time only when the number of items is small. Thus, acceptable accuracy is only achieved for simple tasks. Thirdly, if the problem is solved for a sufficiently representative set of parameters, this can allow estimating a solution for the entire set of parameters. At the same time, computational complexity increases many times over, and we cannot be sure that the chosen set of parameters is sufficiently representative.

The neural network approach can also be applied to a problem with parameters. In this case, the parameters are included in the number of inputs of the neural network [29]. We have successfully solved a number of similar tasks [15,16]. A modification of the neural network approach to problem (20) consists of finding an approximate solution in the form of the output of an artificial neural network of a given architecture as sum

$$u_N(\mathbf{x}, \mathbf{r}) = \sum_{i=1}^{N} c_i v_i(\mathbf{x}, \mathbf{r}, \mathbf{a}_i), \tag{23}$$

where weights $c_i, \mathbf{a}_i$ are determined in the process of step-by-step network training based on minimizing the loss function of the form $J = J_1 + \delta J_2$. Here,

$$J_1 = \sum_{j=1}^{M} (A(\mathbf{r}_j) u_N(\xi_j, \mathbf{r}_j) - g(\xi_j, \mathbf{r}_j))^2 \tag{24}$$

and

$$J_2 = \sum_{k=1}^{K} (B(\mathbf{r}'_k) u_N(\xi'_k, \mathbf{r}'_k) - f(\xi'_k, \mathbf{r}'_k))^2 \tag{25}$$

Regeneration of points is performed as follows: first, $\mathbf{r}_j$ and $\mathbf{r}'_k$ are re-generated from a parameter change domain and then $\xi_j \in \Omega = \Omega(\mathbf{r}_j)$ and $\xi'_j \in \Gamma = \Gamma(\mathbf{r}_j)$.

If additional data are available, for example, measurements, $J_3 = \sum_{l=1}^{L} (u_N(\xi''_l, \mathbf{r}''_l) - u_l)^2$ is added to the loss function. Here, u_l is a measurement corresponding to the value of a required solution at point ξ''_l and parameter value $\mathbf{r}''_l$.

Despite the natural ability of neural network models to adapt to data, the process of constructing them often has a high computational complexity.

3.2. Parametric Physics-Based Neural Network Approach with FEM Smoothing

In order to combine the advantages of the FEM and neural network method for problems with a parameter, we propose the following hybrid method. Suppose we have constructed a finite element solution for a set of parameter values

$$u_N(\mathbf{x}, \mathbf{r}_j) = \sum_{i=1}^{N} c_{i,j} v_i(\mathbf{x}, \mathbf{r}_j),\ j = 1, \ldots, M. \tag{26}$$

Further, we construct neural network approximations for the coefficients c_i

$$c_i(\mathbf{r}) = \sum_{k=1}^{n} d_{k,i} \varphi_k(\mathbf{r}, \mathbf{a}_{k,i}), \tag{27}$$

where $\varphi_k(\mathbf{r}, \mathbf{a})$ are neural network basis functions. We can apply the previously discussed RBF or perceptron-type functions. Our experience [10,29] shows that the latter perform significantly better in these tasks. Weights (parameters) of all neural networks are selected by minimizing corresponding loss functions, for example, in the form $\sum_{j=1}^{M} (c_{i,j} - \sum_{k=1}^{n} d_{k,i} \varphi_k(\mathbf{r}_j, \mathbf{a}_{k,i}))^2$. The obtained solution

$$u_N(\mathbf{x}, \mathbf{r}) = \sum_{i=1}^{N} \sum_{k=1}^{n} d_{k,i} \varphi_k(\mathbf{r}, \mathbf{a}_{k,i}) v_i(\mathbf{x}, \mathbf{r}) \tag{28}$$

can be utilized not only for interpolation in the range of parameters for which the FEM was used. It is also applicable for extrapolation to the region in which the FEM solution is difficult due to the ill-conditioned matrix.

3.3. Parametric Physics-Based Multilayer Network Architecture Method

As mentioned above, in the multilayer method physics defines network architecture. This technique can be applied to solving parameterized problems virtually without any modifications.

Let us consider the Cauchy problem of the form

$$\begin{cases} \mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x), \mathbf{r}); \\ \mathbf{y}(t(\mathbf{r})) = \mathbf{y}_0(\mathbf{r}); \end{cases} \tag{29}$$

and apply the same recurrence formulas to it as for the problem without a parameter. As a result, we have an approximate solution $\mathbf{y}_n(x, \mathbf{r})$. A separate issue is the selection of dependencies $t(\mathbf{r})$. In simple problems we can use, for example, a linear dependence on $\mathbf{r}$. In tasks with measurements, we can select this dependence in the form of a neural network. In this case, $t(\mathbf{r}) = \sum_{k=1}^{n} p_k \varphi_k(\mathbf{r}, \mathbf{a}_k)$ and $\mathbf{y}_0(\mathbf{r}) = \sum_{k=1}^{n} \mathbf{q}_k \varphi_k(\mathbf{r}, \mathbf{b}_k)$.

At the first stage, we again write down the formula for approximate solution $\mathbf{y}_n(x, \mathbf{r}, t(\mathbf{r}), \mathbf{y}_0(\mathbf{r}))$. At the second stage, we adjust parameters p_k, $\mathbf{a}_k$, $\mathbf{q}_k$ and $\mathbf{b}_k$ by minimizing the loss function $\sum_{l=1}^{L} ||\mathbf{y}_n(\xi_l'', \mathbf{r}_l, t(\mathbf{r}_l), \mathbf{y}_0(\mathbf{r}_l)) - \mathbf{y}_l||^2$.

The study of the ability of a multilayer model to learn upon receipt of additional information about an object has been started in [37]. In [8], an arbitrarily accurate implicit Runge–Kutta time stepping scheme with an unlimited number of stages is utilized as the basis for one of two algorithms for constructing data-driven models based on physical constraints described by general non-linear partial differential equations. In addition, the problem of the data-driven discovery of partial differential equations is solved. However, the authors acknowledge the complexity of the proposed algorithms in comparison with classical numerical methods.

4. Results

We have studied the application of methods described above for solving the benchmark parameterized boundary value problem [33]

$$\begin{cases} u''(x) + a(s)u'(x) + b(s)u(x) = 0; \\ u(0) = 0, u(1) = 1; \end{cases} x \in [0,1]. \tag{30}$$

with coefficients $a(s) = 1/s + s$ and $b(s) = 1/s - s$.

For small values of s, problem (30) is stiff and can be regarded as a parameterized singular perturbation problem with a boundary layer in the vicinity of zero. This problem has an analytical solution that allows comparing the results of different methods clearly.

4.1. Physics-Based Neural Network Approach

Let us apply the neural network technique to solving Problem (30). The task is solved for fixed values of parameter s; a solution also has the form (9). The weights of the neural network are selected by minimizing the loss function

$$\sum_{i=1}^{m} (u''(x_i) + a(s)u'(x_i) + b(s)u(x_i))^2 + \delta_1 u^2(0) + \delta_2 (u(1) - 1)^2. \tag{31}$$

Test points $\{x_i\}$ are randomly chosen on the interval $[0,1]$ with regeneration every 5 steps of the optimization process. RProp is used as a minimization algorithm [38]. It is known [39,40] that neural networks with one hidden layer are universal approximators. With regard to the solution accuracy control, there are two different options. The first one is for setting the threshold value for the optimized loss function and the second one is for setting the limit on the number of network learning epochs. In the second case, among the results obtained, the one for which the loss function has the smallest value is selected. In addition, the residual based on real data about the object can be taken into account.

To solve the problem, we use two types of basis functions, sigmoids and Gaussians, and a different number of hidden layer neurons. For each neural network solution the plot of training loss history is presented. One epoch corresponds to 100 training point regenerations. Between each two regenerations, 5 steps of the RProp algorithm are conducted. The number of training points at each step equals 100. Figure 1 represents a solution to (30) for fixed $s = 0.1$ in the case of a Gaussian neural network with 5 neurons in the hidden layer. The correspondent error is shown in Figure 2. The analytical solution has the form

$$\begin{aligned} \mathrm{nn}_5(x) = {} & 0.6934e^{-1.0151(-0.9445+x)^2} + 0.5966e^{-30.8231(0.0819+x)^2} + \\ & 0.4256e^{-63.199(0.1164+x)^2} + 2.1744e^{-1.3991(0.1815+x)^2} - 46.4433e^{-8.6259(0.5628+x)^2}. \end{aligned} \tag{32}$$

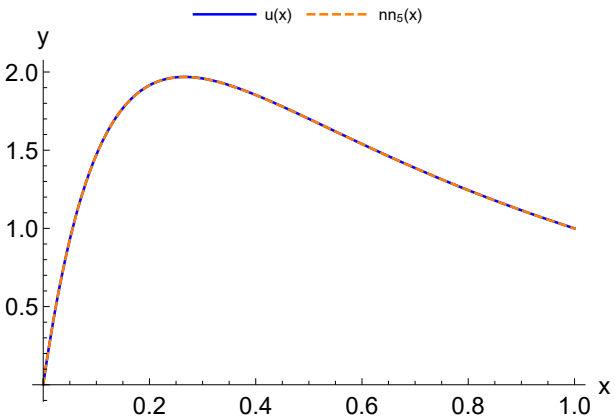

Figure 1. Plots of neural network solution $\mathrm{nn}_5(x)$ (9) (number of layers $n = 5$; Gaussian activation functions) as applied to problem (30) and its exact solution $u(x)$ (parameter value $s = 0.1$), corresponding to dashed and solid lines, respectively.

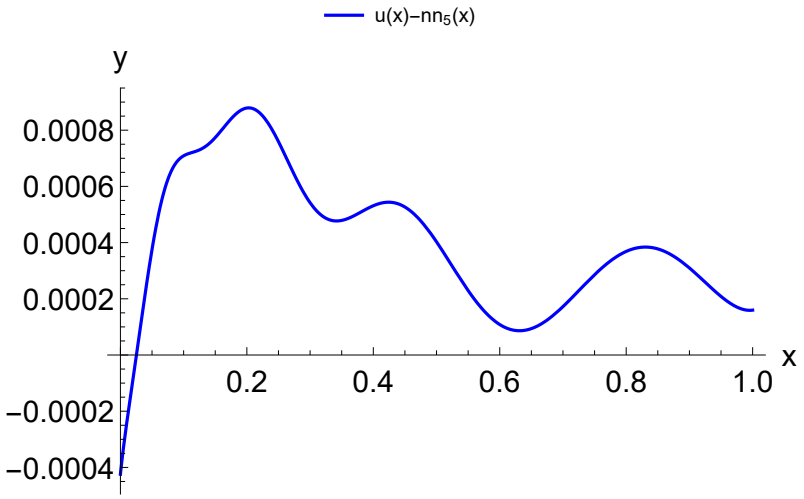

Figure 2. Plot of the error of neural network solution $\mathtt{nn}_5(x)$ (9) (number of hidden layer neurons $n = 5$, Gaussian activation functions) as applied to problem (30) (parameter value $s = 0.1$).

The training loss history is shown in Figure 3.

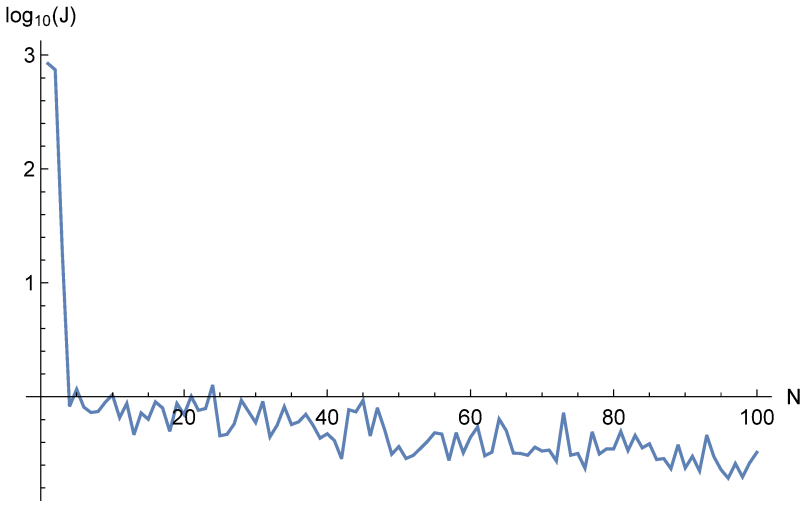

Figure 3. Plot of training loss J history of neural network solution $\mathtt{nn}_5(x)$ (9) to problem (30), parameter value $s = 0.1$, with respect to number N of training epochs. The number of hidden layer neurons $n = 5$; Gaussian activation functions.

For sigmoid basis functions, the solution, and error plots are shown in Figures 4 and 5. The analytical solution, in this case, looks like

$$\begin{aligned}\mathtt{n}_5(x) &= 2.9401\mathtt{tanh}(0.1548(-1291.23+x)) - 0.2423\mathtt{tanh}(2.8222(-0.5127+x))\\ &- 3.4427\mathtt{tanh}(0.2677(-0.4512+x)) + 0.8904\mathtt{tanh}(6.6248(-0.0469+x))\\ &+ 3.7422\mathtt{tanh}(9.4914(0.0895+x)).\end{aligned} \tag{33}$$

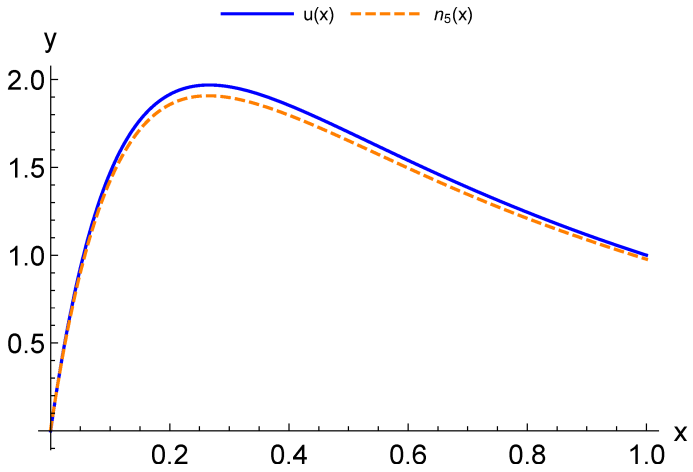

Figure 4. Plots of neural network solution $\mathtt{n}_5(x)$ (9) (number of layers $n = 5$; sigmoid activation functions) as applied to problem (30) and its exact solution $u(x)$ (parameter value $s = 0.1$), corresponding to dashed and solid lines, respectively.

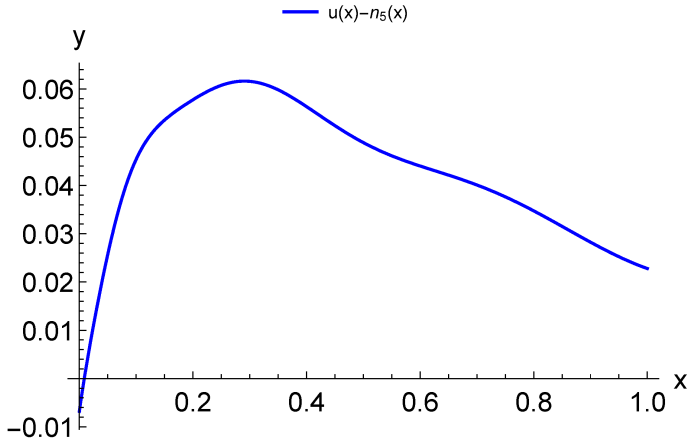

Figure 5. Plot of the error of the neural network solution $\mathtt{n}_5(x)$ (9) (number of layers $n = 5$; sigmoid activation functions) as applied to problem (30), parameter value $s = 0.1$.

Figure 6 illustrates the training loss history for solution (33) as the decimal logarithm with respect to the training epoch.

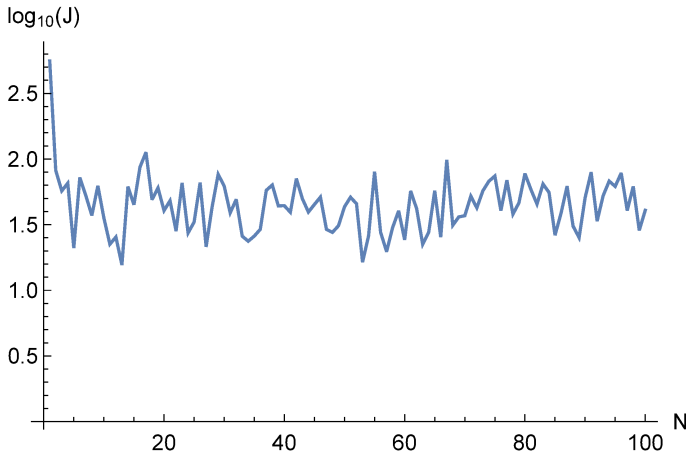

Figure 6. Plot of training loss J history of neural network solution $\mathtt{nn}_5(x)$ (9) to problem (30), parameter value $s = 0.1$, with respect to number N of training epochs. Number of hidden layer neurons $n = 5$; sigmoid activation functions.

For a problem in a stiff statement, for example, at $s = 0.01$, without additional effort, it is possible to obtain an acceptable solution only using a network with 20 hidden layer neurons and sigmoid basis functions. The corresponding plots of the approximation and its error are shown in Figures 7 and 8.

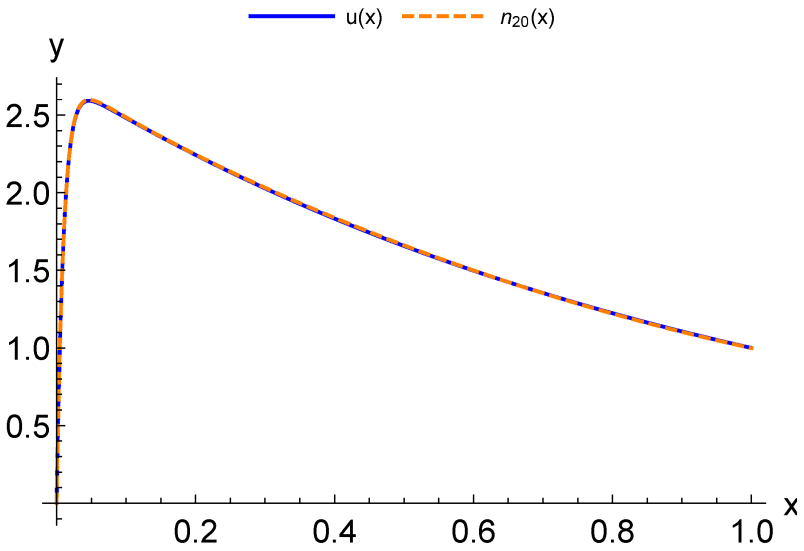

Figure 7. Plots of neural network solution $\mathrm{n}_{20}(x)$ (9) (number of layers $n = 20$; sigmoid activation functions) as applied to problem (30) and its exact solution $u(x)$ (parameter value $s = 0.01$), corresponding to dashed and solid lines, respectively.

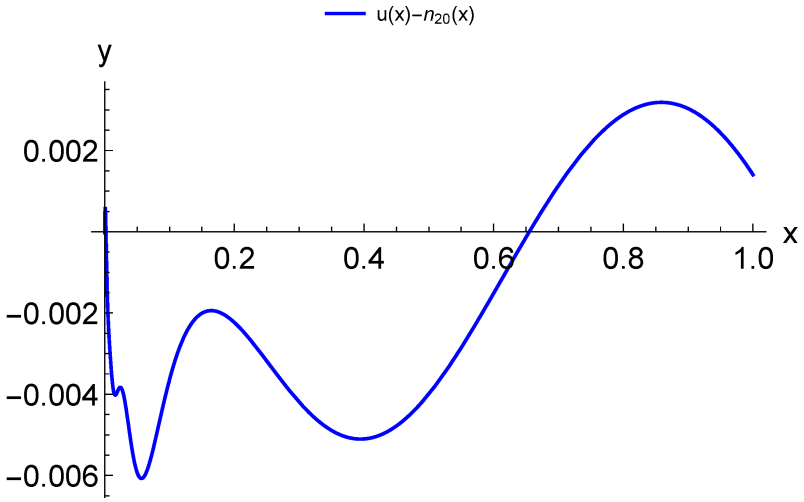

Figure 8. Plots of the error of neural network solution $\mathrm{n}_{20}(x)$ (9) (number of layers $n = 20$; sigmoid activation functions) as applied to problem (30), parameter value $s = 0.01$.

In this case, the neural network solution has a more cumbersome form. Figure 9 shows the training loss history.

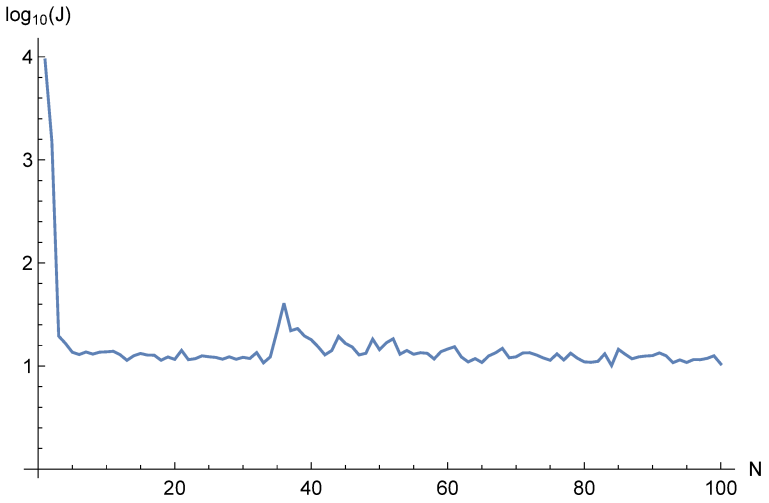

Figure 9. Plot of training loss J history of neural network solution $\mathtt{nn}_{10}(x)$ (9) to problem (30), parameter value $s = 0.01$, with respect to number N of training epochs. Number of hidden layer neurons $n = 20$; sigmoid activation functions.

In [41,42], physics-informed deep neural networks with adaptive activation functions are presented. Activators allow increasing the learning rate of neural networks. We have considered a network with one hidden layer and a global adaptive Gaussian activation function. However, the dynamic change in the loss function topology has not been observed.

The approximation of a parametric solution by a neural network requires a long training process. For a small number of layers, a result with a large approximation error can be obtained.

Based on the results for various basis functions in the case of the fixed-parameter problem statement, we have chosen a perceptron network with 100 hidden layer neurons and sigmoid activation functions to build a parametric neural network solution. Test points $\{x_i, s_i\}$ are randomly chosen on the domain $[0, 1] \times [0.03, 0.1]$. RProp is used as a minimization algorithm [38]. One epoch corresponds to 10 training point regenerations. Between each two regenerations, 5 steps of the RProp algorithm are conducted. The number of training points at each step equals 200.

Figure 10 illustrates the training loss history.

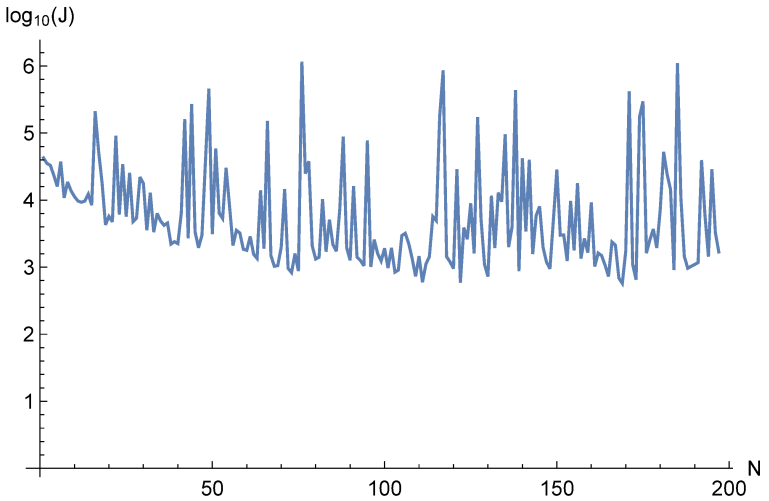

Figure 10. Plot of training loss J history of neural network solution $\mathtt{nn}_{100}(x, s)$ (9) to problem (30), parameter value $s = 0.01$, with respect to number N of training epochs. Number of hidden layer neurons $n = 20$; sigmoid activation functions.

A 3D plot of the approximation error is shown in Figure 11.

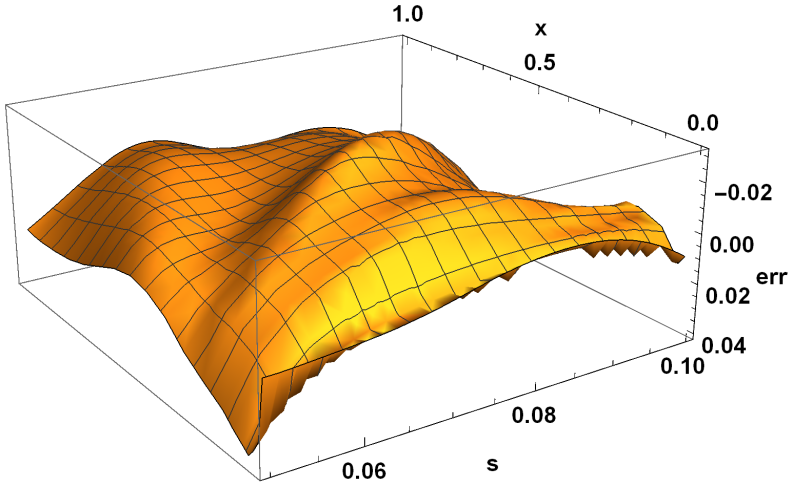

Figure 11. 3D plot of error of neural network parametric solution $\mathtt{nn}_{100}(x, s)$ (9) to problem (30) with respect to $x \in [0, 1]$ and $s \in [0.05, 0.1]$. Number of hidden layer neurons $n = 100$; sigmoid activation functions.

4.2. Physics-Based Neural Network Approach with FEM Smoothing

To solve the parameterized problem, we use a neural network approach to smooth the FEM solutions constructed for fixed values of the parameter. The peculiarities of our method require utilizing a single set of FEM basis functions regardless of the parameter value. Thus, we do not use a FEM with adaptive grids (adaptive basis functions).

To start, we have solved system (30) by using the FEM with different values of a number K of equidistant vertexes x_k and various values of a parameter s. For simplicity, instead of $u(x)$, we have sought an approximate solution to the problem with homogeneous boundary conditions, i.e., an approximation for $u(x) - x$.

It is the one-dimensional case, so an approximate solution to this system takes the form

$$u_N(x) = \mathbf{c}(s) \cdot \mathbf{u}, \tag{34}$$

where vector $\mathbf{u}$ consists of K piecewise linear functions $u_k(x)$

$$u_k(x) = \begin{cases} \dfrac{x - x_{k-1}}{x_k - x_{k-1}}, \ x \in [x_{k-1}, x_k]; \\ \dfrac{x_{k+1} - x}{x_{k+1} - x_k}, \ x \in [x_k, x_{k+1}]; \\ 0, \ \text{otherwise}; \end{cases} \tag{35}$$

the coefficients $\mathbf{c}(s) = \{c_k(s)\}$ are obtained by solving the corresponding systems of linear equations.

Thus, we have calculated the values of coefficients $c_i(s_l)$ of FEM solutions (34) for $s_l = 0.05, \ldots, 0.1$ with step 0.01 and $K = 10$ nodes. In the end, we have obtained training sets $\{s_l, c_i(s_l)\}$ which are utilized for the neural network approximations of FEM coefficients depending on parameter s

$$p_i(s) = a_{1i} + a_{2i}\mathtt{tanh}(a_{3i}(s - a_{4i})). \tag{36}$$

After approximating each of the coefficients, we have a general parametric solution

$$\mathtt{znn}(s, x) = \sum_{i=1}^{K-1} p_i(s)\varphi_i(x). \tag{37}$$

This solution, in turn, can be considered a neural network output with piecewise-specified activation functions

$$\varphi_i(x) = \begin{cases} 1 - |K(x - \frac{i}{K})|, \ |x - \frac{i}{K}| < \frac{1}{K}; \\ 0, \ \text{otherwise.} \end{cases} \tag{38}$$

The neural network (37) is a hybrid version of an RBF network with preselected centers and widths and a perceptron with one hidden layer.

Figure 12 represents neural network extrapolations of the solution to problem (30) for parameter s value 0.01, which is not included in the training set. Parametric solution (37) obtained in this way shows a fairly good approximation of a solution to problem (30) for small values of parameter s. In order to compare, Figure 12 shows the FEM solution with the same number $K = 20$ of non-adaptive basis functions (35), which reflects the stiffness of problem (30) at small values of the parameter. In addition, simple smoothing of this FEM solution, from which the term x is subtracted, by a network with 3 hidden layer neurons

$$zn(x, \mathbf{p}, \mathbf{q}, \mathbf{r}) = p_0 + \sum_{i=1}^{3} p_i \mathtt{tanh}(q_i(x - r_i)), \tag{39}$$

also improves the quality of the solution (see Figure 12, red and orange lines).

The parameters $\mathbf{p}$, $\mathbf{q}$, $\mathbf{r}$ are selected by optimizing the standard loss function using the RProp method.

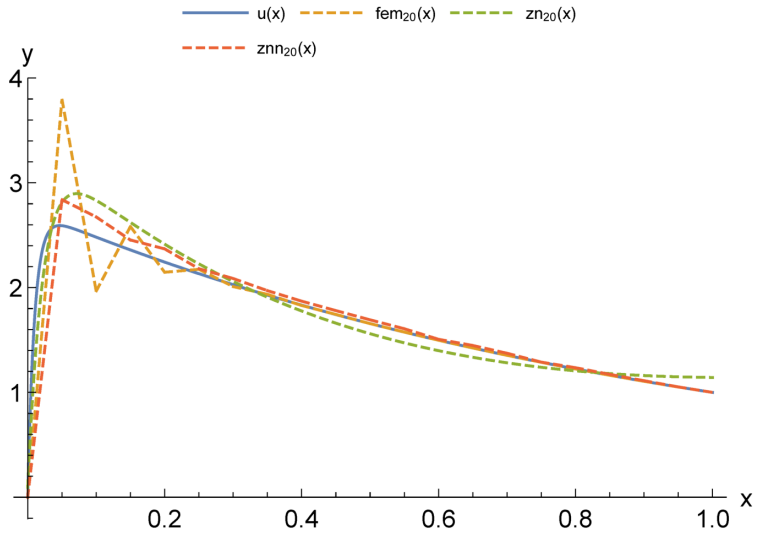

Figure 12. Plots of parametric FEM neural network solution $\mathtt{znn}_{20}(x)$ (37) as applied to parameterized problem (30), FEM solution $\mathtt{fem}_{20}(x)$ and neural network FEM solution smoothing $\mathtt{zn}_{20}(x)$ (39) as applied to problem (30) (parameter value $s = 0.01$; number of elements $K = 20$), and its exact solution $u(x)$, corresponding to dashed lines and solid line, respectively.

4.3. Physics-Based Multilayer Network Architecture Method

To apply an idea described in subsection Physics-Based Multilayer Network Architecture Method, we represent a solution of system (30) as a linear combination of two

boundary problem solutions $\varphi(x) = (\varphi_1(x), \varphi_2(x))$ and $\xi(x) = (\xi_1(x), \xi_2(x))$. The first problem is

$$\begin{cases} y'(x) = z(x); \\ z'(x) = -a(s)z(x) - b(s)y(x), \ x \in [0,1]; \\ y(x_0) = 1; \\ z(x_0) = 0. \end{cases} \tag{40}$$

and the second one is

$$\begin{cases} y'(x) = z(x); \\ z'(x) = -a(s)z(x) - b(s)y(x), \ x \in [0,1]; \\ y(x_0) = 0; \\ z(x_0) = 1. \end{cases} \tag{41}$$

Here, optimal point x_0 depends on a parameter s and can be selected by a researcher. Throughout this article, an optimal point is considered a point minimizing the maximum error of the approximate solution on the segment $[0,1]$.

Solutions to Problems (40) and (41) can be obtained by means of the multilayer method. Then a solution of system (30) is

$$u(x) = c_1\varphi_1(x) + c_2\xi_1(x). \tag{42}$$

Coefficients $c_{1,2}$ are determined by the boundary conditions of system (30)

$$\begin{cases} c_1\varphi_1(0) + c_2\xi_1(0) = 0; \\ c_1\varphi_1(1) + c_2\xi_1(1) = 1. \end{cases} \tag{43}$$

To solve problems (40) and (41), different types of function F (see (13)) have been considered. In the case of using the implicit Euler method (14) as a basis for a multilayer solution, systems (40) and (41) are transformed to

$$\begin{cases} y'_k(x) = y_{k-1} + hz_k; \\ z_k = z_{k-1} - h(a(s)z_k + b(s)y_k). \end{cases} \tag{44}$$

This system can be solved with respect to y_k and z_k. As a starting point x_0, we considered 0 and the optimal point. The resulting multilayer solution to problem (30) for $n = 10$ layers and optimal point $x_0 = 0.151$ has the analytical form

$$y_n(x, 0.151, 10) = \frac{58.9 + 303x + 850x^2 + 1940x^4 + 1680x^5 + 100x^6 + 391x^7 + 90.6x^8 + 9.44x^9}{10^{-9}x^{-1}(8.59 + 9.9x + x^2)^{10}}. \tag{45}$$

Corresponding multilayer solutions based on the implicit Euler method for $x_0 = 0$ and $x_0 = 0.151$ at $s = 0.1$ are shown in Figure 13.

If function F from (13) corresponds to the trapezoidal method, systems (40) and (41) are reduced to the form

$$\begin{cases} y_k = y_{k-1} + \dfrac{1}{2}h(z_k + z_{k-1}); \\ z_k = z_{k-1} - \dfrac{1}{2}h(a(s)z_k + b(s)y_k + a(s)z_{k-1} + b(s)y_{k-1}). \end{cases} \tag{46}$$

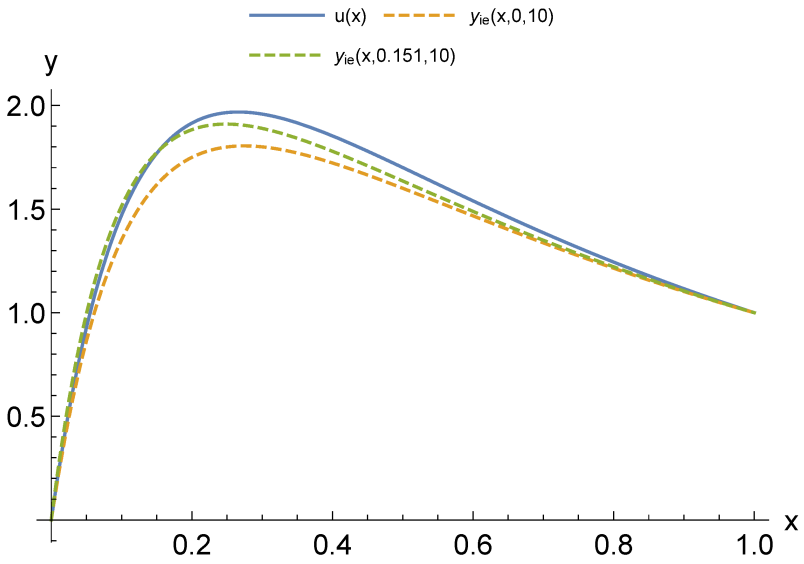

Figure 13. Plots of multilayer solutions $y_{\mathtt{ie}}(x, 0, 10)$, $y_{\mathtt{ie}}(x, 0.151, 10)$ (42) based on the implicit Euler method (number of layers $n = 10$) as applied to problem (30) and its exact solution $u(x)$ (parameter value $s = 0.1$), corresponding to dashed lines and solid line, respectively.

At the same time, for parameter $s = 0.1$, a three-layer network gives a solution with an error less than 0.02 even without selecting the optimal point. This fact is illustrated in Figure 14. It also shows how the selection of the starting point has a significant effect on a two-layer network result.

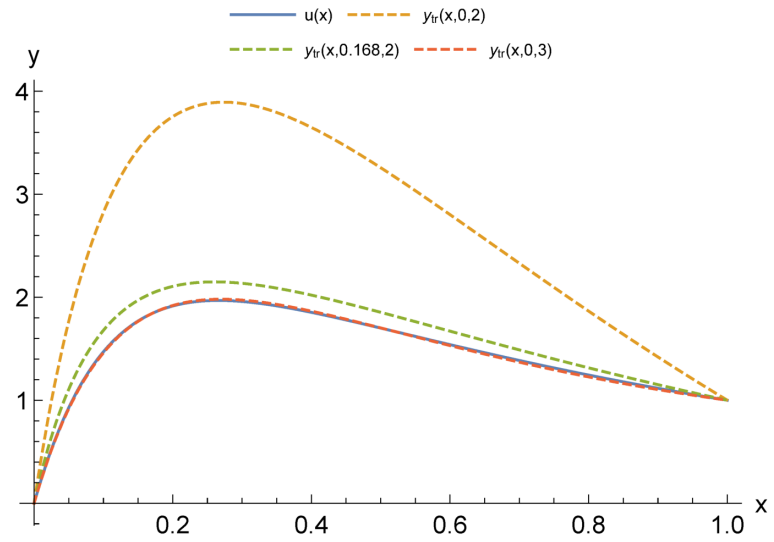

Figure 14. Plots of multilayer solutions $y_{tr}(x, 0, 2)$, $y_{tr}(x, 0.168, 2)$, $y_{tr}(x, 0, 3)$ (number of layers $n = 2, 2, 3$) as applied to problem (30) and its exact solution $u(x)$ (parameter value $s = 0.1$), corresponding to dashed lines and solid line, respectively.

Let us consider a more stiff formulation of the problem when the parameter s value is equal to 0.001. In this case, it is possible to obtain a sufficiently accurate approximate solution by using a two-layer network by selecting the optimal point (see Figure 15), and Formula (13) leads to a simple form of the solution

$$y_{\mathtt{tr}}(x, 0.0046, 2) = \frac{x(-0.0498 + 44.7x - 22.5x^2 + 2.79x^3)}{(0.00238 - 3.99x - x^2)^2}. \tag{47}$$

Increasing the number of layers maintains the effect.

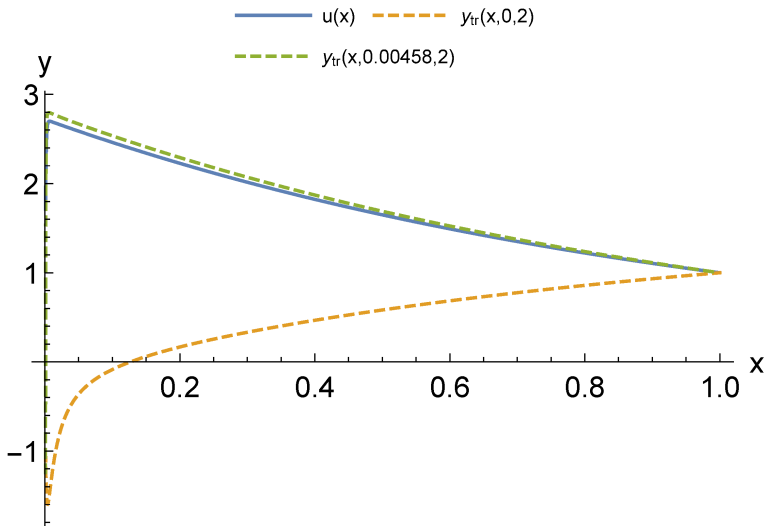

Figure 15. Plots of multilayer solutions $y_{tr}(x,0,2)$, $y_{tr}(x,0.0048,2)$ (number of layers $n=2$) as applied to problem (30) and its exact solution $u(x)$ (parameter value $s=0.001$), corresponding to dashed lines and solid line, respectively.

Let us solve parameterized problem (30) by a multilayer method. In this case, parameter s is naturally contained in a solution,

$$y(x,s) = c_1(s)\varphi_1(s,x) + c_2(s)\xi_1(s,x). \tag{48}$$

Thus, a one-layer solution based on the trapezium method and for $x_0 = 1.7s$ has the form

$$y_1(x,s) = \frac{x(-1.04+0.969s-1.35s^2+s^4)(0.059x+s^3-0.56s^2x-s-1.39)(s^3-0.59s^2-s-0.796)^{-1}}{s^4+1.18(1-x)s^3-(0.692x+0.346x^2)+s(1.18x-0.208)+s^2(0.346x^2-0.692x-1)} \tag{49}$$

In this case, the point x_0, as mentioned earlier, depends on s but should not be beyond the considered interval.

Despite problem (30) being stiff for small s, we obtain a fairly good solution for all values of s from the interval $[0.001, 0.1]$ using the multilayer technique based on the trapezium method with 8 layers. This is confirmed by the 3D plot of the approximation error in Figure 16.

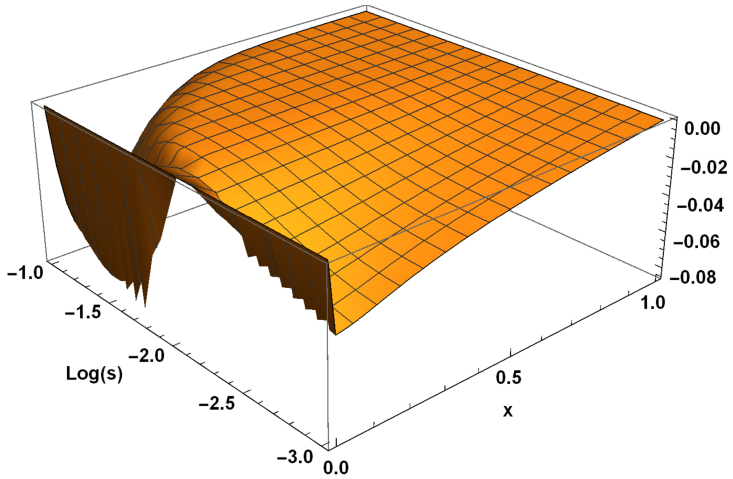

Figure 16. 3D plot of the error of the multilayer parametric solution $y_8(x,s)$ (48) (trapezium method; number of layers $n=8$; initial point $x_0 = \max(5s,1)$) as applied to problem (30), with respect to $x \in [0,1]$ and $\log(s)$, $s \in [0.1, 0.001]$.

Figure 17 illustrates that choosing the required number of layers allows us to obtain an approximation of any accuracy.

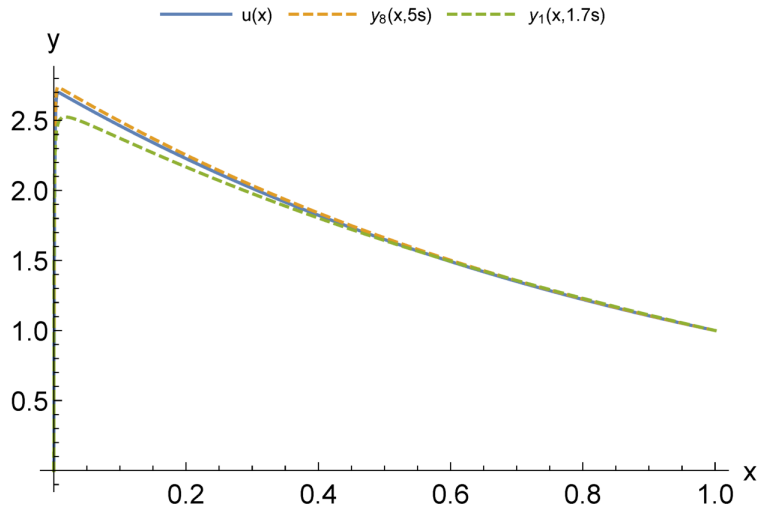

Figure 17. Plots of multilayer parametric solutions $y_8(x, 5s)$, $y_1(x, 1.7s)$ with optimal initial points, as applied to problem (30), and its exact solution $u(x)$ (parameter value $s = 0.001$), corresponding to dashed lines and solid line, respectively.

The above results of the multilayer method in solving a parameterized problem singularly perturbed by a small parameter make it possible to consider this method as promising in the modeling and control of complex systems. It is important to note that, if the problem of constructing a preliminary model of a real object is considered, a fairly good approximation for small values of the parameter gives a solution with a single layer. In addition, the known estimates of approximation errors of the basic recurrence relations can be used to define the number of layers to obtain a solution with a preselected accuracy. The construction of a multilayer solution takes little time and can immediately be used for different values of the parameter. Unlike various neural network approaches, a multilayer model does not require training. At the same time, the subsequent adjustment of the parameters of such a solution using the apparatus of neural networks is not excluded in the case of additional data appearing. On this occasion, the general neural network approach should show itself well, which is confirmed by both the results of our previous article [15] and the hybrid neural network models constructed in this work using the "poor" results of the classical FEM.

In the context of new challenges facing the modern control theory and, in particular, the methodology for creating CPS, problems are often ill-posed and poorly formalized, and are set using various heterogeneous data, parameterized differential equations of unclear structure, or without them. Requirements for the speed of processing such data and solving dynamic equations are growing. Thus, the development of versatile low-cost methods presented in this work is an important challenge.

5. Discussion

For modeling CPS, an urgent issue is the transition from differential models to functional ones. At the same time, an important factor is a possibility to adapt these models according to the observation data of the object. Our computational experiments allow drawing preliminary conclusions about what kind of models may be more suitable for further adaptation.

The main advantage of finite element models is that they are well studied. Accordingly, there are a large number of software packages and libraries. At the same time, with the appearance of stiffness in the problem being solved, the number of necessary functions to

obtain acceptable accuracy increases sharply. Moreover, difficulties in adapting the solution increase, since it becomes unstable against data errors. This problem is aggravated by the local nature of the basis functions. Neural network models are more capable of adaptation. However, the process of their construction has a high computational complexity that also greatly increases with the appearance of stiffness. This situation allows us to look prospectively at the hybrid approaches we have considered. Even more interesting are multilayer models, which are the most compact for a given accuracy. We have already studied their adaptive properties on one problem [37]. We intend to continue this study in further research.

Another important issue is computational costs. The fastest are multi-layered methods as they do not require training. The next ones are methods with constant basis (activation) functions but, as a result of their application, larger models are obtained. It also takes more time to adapt them further. The training of neural networks can be accelerated, but this is relevant for the stage of adaptation from measurements in real time. In the future, we intend to compare the adaptive properties of multilayer models and neural networks with different acceleration options: structure adaptation, faster methods, and the introduction of hyperparameters.

The issue of the extension of the presented results to nonlinear problems is worth specific discussion. For these problems, the advantages of the FEM are substantially leveled, since the construction of a FEM solution is no longer reduced to solving a linear system. At the same time, the complexity of building a neural network and a multilayer solution changes only slightly.

Author Contributions: Conceptualization, T.L., G.M. and D.T.; methodology, T.L., G.M. and D.T.; software, D.T.; validation, T.L., G.M. and D.T.; formal analysis, T.L., G.M. and D.T.; investigation, T.L., G.M. and D.T.; resources, T.L., G.M. and D.T.; data curation, T.L., G.M. and D.T.; writing—original draft preparation, T.L., G.M. and D.T.; writing—review and editing, T.L., G.M. and D.T.; visualization, T.L.; supervision, T.L., G.M. and D.T.; project administration, T.L., G.M. and D.T.; funding acquisition, T.L., G.M. and D.T. All authors have read and agreed to the published version of the manuscript.

Funding: The research was partially funded by the Ministry of Science and Higher Education of the Russian Federation as part of the World-Class Research Center program: Advanced Digital Technologies (contract No. 075-15-2020-934 dated 17.11.2020).

Data Availability Statement: Data available on request from the authors.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CPS	Cyber-physical systems
DT	Digital twin
ML	Machine learning
FEM	Finite element method

References

1. Lee, J.; Bagheri, B.; Kao, H. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manuf. Lett.* **2015**, *3*, 18–23. [CrossRef]
2. Rasheed, A.; San, O.; Kvamsdal,T. Digital Twin: Values, Challenges and Enablers From a Modeling Perspective. *IEEE Access* **2020**, *8*, 21980–22012. [CrossRef]
3. RAI, R.; Sahu, C.K. Driven by Data or Derived through Physics? A Review of Hybrid Physics Guided Machine Learning Techniques with Cyber-Physical System (CPS) Focus. *IEEE Access* **2020**, *8*, 71050–71073. [CrossRef]
4. Frank, M.; Drikakis, D.; Charissis, V. Machine-Learning Methods for Computational Science and Engineering. *Computation* **2020**, *8*, 15. [CrossRef]
5. Kovalchuk, S.V.; Metsker, O.G.; Funkner, A.A.; Kisliakovskii, I.O.; Nikitin, N.O.; Kalyuzhnaya, A.V.; Vaganov, D.A.; Bochenina, K.O. A Conceptual Approach to Complex Model Management with Generalized Modelling Patterns and Evolutionary Identification. *Complexity* **2018**, *2018*, 5870987. [CrossRef]

6. Kim, S.W.; Kim, I.; Lee, J.; Lee, S. Knowledge Integration into deep learning in dynamical systems: An overview and taxonomy. *J. Mech. Sci. Technol.* **2021**, *35*, 1331–1342. [CrossRef]
7. Zhang, Z.; Rai, R.; Chowdhury, S.; Doermann, D. MIDPhyNet: Memorized infusion of decomposed physics in neural networks to model dynamic systems. *Neurocomputing* **2021**, *428*, 116–129. [CrossRef]
8. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, 378, 686–707. [CrossRef]
9. Lagaris, I.E.; Likas, A.; Fotiadis, D.I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* **1998**, *9*, 987–1000. [CrossRef]
10. Tarkhov, D.; Vasilyev, A. New neural network technique to the numerical solution of mathematical physics problems. II: Complicated and nonstandard problems. *Opt. Mem. Neural Netw. (Inf. Opt.)* **2005**, *14*, 97–122.
11. Jia, X.; Willard, J.; Karpatne, A.; Read, J.; Zwart, J.; Steinbach, M.; Kumar, V. Physics guided RNNs for modeling dynamical systems: A case study in simulating lake temperature profiles. In Proceedings of the SIAM International Conference on Data Mining, SDM19, Calgary, AB, Canada, 2–4 May 2019; pp. 558–566.
12. Dallas, S.; MacHairas, K.; Papadopoulos, E. A Comparison of Ordinary Differential Equation Solvers for Dynamical Systems with Impacts. *J. Comput. Nonlinear Dyn.* **2017**, *12*, 6. [CrossRef]
13. Benner, P.; Gugercin, S.; Willcox, K. A Survey of Projection-Based Model Reduction Methods for Parametric Dynamical Systems. *SIAM Rev.* **2015**, *57*, 483–581. [CrossRef]
14. Muhanna, R.L.; Mullen, R.L.; Zhang, H. Penalty-based solution for the interval finite-element methods. *J. Eng. Mech.* **2005**, *131*, 1102–1111. [CrossRef]
15. Lazovskaya, T.; Tarkhov, D. Fresh approaches to the construction of parameterized neural network solutions of a stiff differential equation. *St. Petersburg Polytech. Univ. J. Phys. Math.* **2015**, *1*, 192–198. [CrossRef]
16. Lazovskaya, T.V.; Tarkhov, D.A.; Vasilyev, A.N. Parametric neural network modeling in engineering. *Recent Pat. Eng.* **2017**, *11*, 10–15. [CrossRef]
17. Famelis, I.T.; Kaloutsa, V. Parameterized neural network training for the solution of a class of stiff initial value systems. *Neural Comput. Appl.* **2020**. [CrossRef]
18. Wang,Y.; Chen, S.; Wu, X. A rational spectral collocation method for solving a class of parameterized singular perturbation problems. *J. Comput. Appl. Math.* **2010**, *233*, 2652–2660.
19. Das, P. Comparison of a priori and a posteriori meshes for singularly perturbed nonlinear parameterized problems. *J. Comput. Appl. Math.* **2015**, *290*, 16–25.
20. Guglielmi,N.; Hairer, E. Solutions leaving a codimension- 2 sliding. *Nonlinear Dyn.* **2017**, *88*, 1427–1439. [CrossRef]
21. Kudu, M. A parameter uniform difference scheme for the parameterized singularly perturbed problem with integral boundary condition. *Adv. Differ. Equ.* **2018**, *170*, 2018. [CrossRef]
22. Shakti, D.; Mohapatra, J. Parameter-Uniform Numerical Methods for a Class of Parameterized Singular Perturbation Problems. *Numer. Anal. Appl.* **2019**, *12*, 176–190. [CrossRef]
23. Liu, H.; Xing, B.; Wang, Z.; Li, L. Legendre Neural Network Method for Several Classes of Singularly Perturbed Differential Equations Based on Mapping and Piecewise Optimization Technology. *Neural Process. Lett.* **2020**. [CrossRef]
24. Lazovskaya, T.; Tarkhov, D. Multilayer neural network models based on grid methods. In *IOP Conference Series: Materials Science and Engineering*; IOP Publishing: Bristol, UK, 2016. [CrossRef]
25. Hughes, T.J.R. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*; Prentice-Hall: Englewood Cliffs, NJ, USA, 1987.
26. Huang, G.; Zhu, Q.; Siew, C. Extreme learning machine: Theory and applications. *Neurocomputing* **2006**, *70*, 489–501. [CrossRef]
27. Huang, G.; Chen, L.; Siew, C. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Trans. Neural Netw.* **2006**, *17*, 879–892. [CrossRef] [PubMed]
28. Li, B.; Li, Y.; Rong, X. The extreme learning machine learning algorithm with tunable activation function. *Neural Comput. Appl.* **2013**, *22*, 531–539. [CrossRef]
29. Tarkhov, D.; Vasilyev, A. *Semi-Empirical Neural Network Modeling and Digital Twins Development*; Academic Press: Cambridge, MA, USA, 2020.
30. Antonov, V.; Tarkhov, D.; Vasilyev, A. Unified approach to constructing the neural network models of real objects. Part 1. *Math. Methods Appl. Sci.* **2018**, *41*, 9244–9251. [CrossRef]
31. Geneva, N.; Zabaras, N. Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks. *J. Comput. Phys.* **2020**, *403*, 109056. [CrossRef]
32. Zhang, Z.; Sun, C. Structural damage identification via physics-guided machine learning: A methodology integrating pattern recognition with finite element model updating. *Struct. Health Monit.* **2020**. [CrossRef]
33. Nayfeh, A.H. *Perturbation Methods*; Hardcover; Wiley-VCH Verlag GmbH: Weinheim, Germany, 1973.
34. Hairer, E.; Wanner, G. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*; Springer: Berlin/Heidelberg, Germany, 1996.
35. Lazovskaya, T.; Malykhina, G.; Tarkhov, D. Construction of an Individual Model of the Deflection of a PVC-Specimen Based on a Differential Equation and Measurement Data. In Proceedings of the 2020 International Multi-Conference on Industrial Engineering and Modern Technologies, FarEastCon 2020, Vladivostok, Russia, 6–9 October 2020. [CrossRef]

36. Beucler, T.; Pritchard, M.; Rasp, S.; Ott, J.; Baldi, P.; Gentine, P. Enforcing Analytic Constraints in Neural Networks Emulating Physical Systems. *Phys. Rev. Lett.* **2021**, 126. [CrossRef]
37. Boyarsky, S.; Lazovskaya, T.; Tarkhov, D. Investigation of the Predictive Capabilities of a Data-Driven Multilayer Model by the Example of the Duffing Oscillator. In Proceedings of the 2020 International Multi-Conference on Industrial Engineering and Modern Technologies, FarEastCon 2020, Vladivostok, Russia, 6–9 October 2020. [CrossRef]
38. Braun, H.; Riedmiller, M. A direct adaptive method for faster backpropagation learning: The Rprop algorithm. In Proceedings of the IEEE International Conference on Neural Networks, San Francisco, CA, USA, 28 March–1 April 1993; pp. 586–591.
39. Hornik, K.; Tinchcombe, M.; White, H. Multilayer Feedforward Networks are Universal Approximators. *Neural Netw.* **1989**, *2*, 359–366. [CrossRef]
40. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control. Signals Syst.* **1989**, *2*, 303–314. [CrossRef]
41. Jagtap, A.D.; Kawaguchi, K.; Em Karniadakis, G. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *J. Comput. Phys.* **2020**, *404*, 109136. [CrossRef]
42. Jagtap, A.D.; Kawaguchi, K.; Em Karniadakis, G. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proc. R. Soc. A* **2020**, *476*, 20200334. [CrossRef] [PubMed]