



Enhanced Training of Physics-Informed Radial Basis Function Networks: Algorithmic Modifications and Applications to Fluid Dynamics

Dmitry Stenkin, Vladimir Gorbachenko*

Department of Computer Technologies, Penza State University, Penza 440026, Russia.

How to cite this paper: Dmitry Stenkin, Vladimir Gorbachenko. (2026) Enhanced Training of Physics-Informed Radial Basis Function Networks: Algorithmic Modifications and Applications to Fluid Dynamics. *International Journal of Statistics and Data Science*, 2(1), 11-29.
DOI: 10.26855/ijds.2026.06.002

Received: March 10, 2026

Accepted: April 26, 2026

Published: May 14, 2026

*Corresponding author: Vladimir Gorbachenko, Department of Computer Technologies, Penza State University, Penza 440026, Russia.

Abstract

This paper presents a comparative study of physics-informed neural networks (PINNs) for solving boundary value problems governed by partial differential equations. Particular attention is given to physics-informed radial basis function networks (PIRBFNs), which exhibit significantly faster convergence than conventional fully connected PINNs, especially when both network weights and radial basis function parameters are optimized simultaneously. Modified training algorithms for PIRBFNs are developed based on the ADAM, Sophia, and Levenberg–Marquardt methods, extended to enable joint optimization of network weights and basis function parameters. This strategy enhances optimization efficiency by improving both convergence speed and the adaptability of the approximation model. Architectural designs and computational procedures are proposed for applying PIRBFNs to a set of benchmark problems in computational fluid dynamics, including Kovasznay flow, the Taylor–Green vortex, and lid-driven cavity flow. Numerical experiments demonstrate that the proposed training algorithms reduce the number of optimization iterations by a factor of 25–41.3 and decrease the total computational time by 24.3–31.0 times compared with the standard Adam algorithm, while maintaining high solution accuracy.

Keywords

Physics-informed neural networks; radial basis function networks; partial differential equations; boundary value problems; scientific machine learning; Navier–Stokes equations; Levenberg–Marquardt method; optimization algorithms; computational hydrodynamics

1. Introduction

The use of neural networks for solving boundary value problems [1, 2] has developed in parallel with the formation of theoretical foundations for neural network methods. Currently, deep neural networks, into whose training mathematical formulations of physical laws are embedded, are actively used to solve boundary value problems described by partial differential equations (PDEs). If the mathematical model of a process is known, then training the network does not require experimental data, which eliminates one of the main problems of supervised learning. Furthermore, the model can be specified approximately, and the neural network allows for its refinement based on available data, for example, by adjusting the equation coefficients within the framework of solving an inverse problem. Training such networks is based on minimizing residuals at a limited set of trial points, taking into account additional information derived from physical laws. This class of methods is called physics-informed neural networks (PINNs).

The modern PINN concept was systematically presented in 2017 [3, 4, 5]. Essentially, PINNs provide a mesh-free approximate analytical solution to a boundary value problem since the result is determined by the network's architecture and parameters. Due to the embedded mathematical model, the solution results possess interpretability. When solving direct boundary value problems, no additional data is required, which implements unsupervised learning and eliminates the overfitting problem associated with noise in data [6].

PINNs are distinguished by their versatility: they are applicable to a wide range of equations with diverse computational domain geometries. They are used in hydrodynamics for modeling turbulent flows [7], in biomechanics — for analyzing soft tissue deformation [8], in climate modeling, materials science, quantum physics [9], and also for solving inverse problems, such as recovering equation parameters from limited or noisy data [10, 11].

The relevance of such models is increasing with the development of cyber-physical systems [12] and the implementation of digital twins [13, 14] — dynamic models that are constantly updated based on sensor data and accurately reflect the state of a physical object. To create a digital twin of a system with distributed parameters, it is necessary to build an adaptive model by solving inverse problems: determining model parameters based on measured object characteristics [15, 16]. Neural network models, particularly PINNs, offer an effective toolkit for solving such tasks.

Modern PINN implementations [17, 18, 19] typically represent deep fully-connected networks with a moderate number of hidden layers (no more than 10 in works [13, 18, 19, 20]), implemented in popular frameworks such as TensorFlow and PyTorch. The key advantages of these frameworks are built-in automatic differentiation [21] and the availability of optimizers designed for training neural networks. Moreover, popular optimizers implement first-order gradient algorithms, which use the gradient vector of the loss function with respect to the network weights and were developed without considering the specifics of PINNs. There are also high-level libraries, for example DeepXDE [18, 22], which simplify specifying equations, boundary conditions, and domain geometry by using low-level operations of the base frameworks.

From a mathematical point of view, solving PDEs using PINNs reduces to an optimization problem with physical constraints. Let's consider solving a PDE in general form [3, 4, 5, 17, 23]. The equation is defined in domain $\Omega \in \mathbb{R}^N$ with boundary $\partial\Omega$

$$f\left(x, t, \frac{\partial u}{\partial t}, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}, L, \Psi\right) = 0, \quad x \in \Omega, t \in [0, T], \quad u(x, t_0) = h(x), \quad x \in \Omega, \quad u(x, t) = g(t), \quad x \in \partial\Omega, t \in [0, T]$$

where $x = (x_1, x_2, \dots, x_N) \in \mathbb{R}^N$ is the spatial coordinate, t is time, f is the function describing the problem with differential operators and parameter Ψ , $u(x, t)$ is the PDE solution with initial $h(x)$ and boundary $g(t)$ conditions.

PINNs approximate the PDE solution as

$$\hat{u}(x, t, \Theta) \approx u(x, t),$$

where Θ is the set of weight and bias vectors of the neural network.

The loss function represents a weighted sum of the squares of the Euclidean norms of the residuals at trial points inside the solution domain, and at the boundary and initial conditions [5]

$$L(\Theta) = \lambda_f L_f(\Theta) + \lambda_{ic} L_{ic}(\Theta) + \lambda_{bc} L_{bc}(\Theta),$$

where

$$L_{ic}(\Theta) = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} \|\hat{u}(x_i, t_0) - h(x_i)\|_2^2, \quad L_{bc}(\Theta) = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} \|u(x_i, t) - g(t)\|_2^2, \quad L_f(\Theta) = \frac{1}{N_f} \sum_{i=1}^{N_f} \|\hat{u}(x_i, t) - g(t)\|_2^2,$$

N_f is the set of trial points inside the solution domain, N_{ic} is the set of trial points for checking initial conditions, N_{bc} is the set of trial points for checking boundary conditions, λ_f , λ_{ic} and λ_{bc} are weights (often $\lambda_f = 1$).

When solving inverse problems, some parameters of the problem are unknown, for example, equation coefficients, initial or boundary conditions. In this case, approximately measured solution values at some points are given. When solving inverse problems, a term accounting for the deviation from known data is added to the loss function. An important advantage of PINNs is a unified approach to solving direct and inverse problems [5, 24, 25]: the network architecture remains unchanged, only the loss function changes.

The process of solving a direct problem includes the following steps:

- generation of trial points (trial points are typically generated randomly);

- creation of a neural network with a specified number of layers, neurons, and activation functions;
- iterative training of the neural network with parameter recalculation until the specified accuracy is achieved;
- obtaining the solution at required points as the output of the trained network.

The theoretical basis of PINNs consists of two propositions:

- use of neural networks as universal function approximators;
- application of a variational approach, where the solution to a boundary value problem is found by minimizing a loss function.

The theoretical basis of PINN is grounded in the universal approximation properties of neural networks. Pioneering work by Cybenko [26], Hornik [28, 29], and others [30] established that feedforward networks can approximate any continuous function on a compact set.

A promising direction for the development of PINNs is the use of physics-informed radial basis function networks (PIRBFN) [5, 15, 2], which are structurally simpler than fully-connected networks and possess more efficient second-order training algorithms. PIRBFNs are better at approximating functions with local features.

PIRBFN [31, 32] consists of two layers. The first layer implements a nonlinear transformation of the coordinate vector of the point at which the solution approximation is computed, using radial basis functions (RBFs). The second layer is a linear weighted sum:

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

where M is the number of radial basis functions, w_m is the weight of the radial basis function φ_m , \mathbf{p}_m is the parameter vector of the radial basis function

The use of PIRBFNs is based on the property of RBFs as universal function approximators. In [33, 34, 35, 36], it has been proven that, given a sufficient number of neurons, RBFs are universal function approximators.

Training a classical radial basis function network includes two stages: at the first, difficult-to-formalize stage, the RBF parameters are chosen; at the second stage, after fixing the RBF parameters, the network output is a linear combination of RBFs, and network training reduces to a linear least squares problem, which guarantees achieving a global minimum in terms of mean squared error (MSE). This distinguishes PIRBFN from fully-connected networks, whose training requires solving a complex non-convex optimization problem. Therefore, with fixed RBF parameters, the radial basis function network gives the best approximation in the space spanned by the chosen basis functions. Radial basis function networks approximate functions with local features better.

Despite their high efficiency, PINNs train slowly. RBFNs offer structural advantages, but their application within PINNs has not been sufficiently studied, particularly for hydrodynamics. Although solutions based on fully connected PINNs are presented in the literature [7, 11, 37, 38], there are known works devoted to the application of radial basis functions for solving the Navier-Stokes equations (see, for example, [39, 40]), there are currently no implementations using PIRBFN for the Navier-Stokes equations. In existing PIRBFN implementations, the RBF parameters are fixed, which limits their adaptability. This work fills this gap.

The aim of this work is the development of efficient training algorithms, architectures, and software implementation of PIRBFN for solving Navier-Stokes equations.

2. Methods

2.1 PIRBFN Training Algorithms

The developed algorithms allow for adjusting not only the network's weight coefficients but also the parameters of the radial basis functions (RBFs). This approach eliminates the need for a complex and poorly formalized stage of manual RBF parameter selection. Joint optimization of these parameters enhances the network's approximation capacity for a given number of neurons and improves asymptotic error estimates, which aligns with the principles of classical approximation theory [35]. In particular, during the optimization process, the RBF centers tend towards an optimal placement, shifting towards regions with more rapid changes in the solution. This allows for more precise consideration of local features of the approximated function. Adaptive placement of centers also minimizes the fill distance and, consequently, reduces the approximation error.

Training PIRBFN with simultaneous adjustment of weights and RBF parameters constitutes a nonlinear optimization problem. However, solving it proves simpler than training fully-connected neural networks because the weights

enter the Hessian linearly, and the nonlinear component of the Hessian has a less complex structure. Gradient-based algorithms for training neural networks were adapted for optimizing the parameters. The adaptation involves forming a unified parameter vector, which includes both the weights and RBF parameters, and computing the gradient of the loss function with respect to each component of this vector.

Gradient algorithms adjust the network's parameter vector by a value proportional to the anti-gradient of the error functional, using a preconditioner:

$$\Theta_{k+1} = \Theta_k - \alpha_k \mathbf{P}_k^{-1} \nabla L(\Theta_k),$$

where Θ_k is the neural network's parameter vector at training step k , α_k is the learning rate, $\nabla L(\Theta_k)$ is the gradient of the error functional for the network parameter vector Θ_k , \mathbf{P}_k is the preconditioning matrix, which modifies the anti-gradient direction to accelerate convergence.

Preconditioning transforms the gradient step to account for the second derivatives (local curvature) of the loss function, thereby implementing a controlled descent along its surface [41, 42]. This approach unifies various optimization algorithms since the choice of preconditioner determines the method's specifics. The preconditioning matrix scales the gradient considering curvature: the step size decreases along directions with high curvature and increases along directions with low curvature.

Let's compare the considered gradient training algorithms from the perspective of using preconditioners. The classical gradient descent algorithm does not use a preconditioner (the preconditioner equals the identity matrix), does not account for curvature, and is characterized by low convergence speed. Modern adaptive methods, such as AdaGrad, RMSProp, Adam, AdaHessian, and others, employ adaptive preconditioning matrices based on moving averages of the second moments of gradients or a diagonal approximation of the Hessian [41].

For second-order gradient algorithms, the optimal preconditioner is the Hessian, used in Newton's method, which perfectly accounts for local curvature and ensures high convergence speed. However, computing the full Hessian is computationally expensive for large models, making Newton's method impractical. Quasi-Newton algorithms construct an approximation of the Hessian. For instance, the Levenberg-Marquardt algorithm uses a preconditioner of the form

$$\mathbf{P} = (\mathbf{J}^T \mathbf{J} + \mu \mathbf{E}) \mathbf{J},$$

where \mathbf{J} is the Jacobian matrix, \mathbf{E} is the identity matrix, μ is a tunable parameter.

Although this algorithm ensures fast convergence, the need to compute the Jacobian matrix and solve a system of linear equations at each step requires significant computational resources, especially for networks with a large number of parameters.

A promising preconditioner is the Sophia algorithm (Second-order Clipped Stochastic Optimization) [43]. As a second-order method, Sophia uses an approximation of the Hessian to calculate the update step. A key feature of Sophia is the approximation of only the Hessian's diagonal, which requires orders of magnitude fewer computations compared to estimating the full Hessian. To enhance stability, Sophia additionally applies a clipping mechanism, scaling the update magnitude for each parameter. As a result, the optimizer is applicable to very large networks and works significantly faster than first-order optimizers. Experiments [43] show Sophia's twofold superiority over AdamW (a modification of Adam) with comparable per-step costs. Despite its potential, Sophia has not been previously applied to training PINNs. In this work, the Sophia algorithm is adapted for training PIRBFN. The adaptation includes:

- Simultaneous optimization of network weights and radial basis function (RBF) parameters, eliminating the need for separate hyperparameter tuning of RBFs.
- A proposed simple diagonal approximation of the Hessian.

Рассмотрим шаги алгоритм Sophia. На каждом шаге t обучения сети выполняются следующие действия. Let's consider the steps of the Sophia algorithm. At each training step, the following actions are performed.

1. Gradient computation. The loss function $L_t(\Theta_t)$ and the gradient vector of the error functional with respect to the network parameters are determined:

$$\mathbf{g}_t = \nabla L_t(\Theta_t),$$

where Θ_t is the network's parameter vector at step, including network weights and RBF parameters.

2. Gradient smoothing using exponential moving average (EMA). To suppress noise while preserving information about local curvature, the exponential moving average (EMA) of the gradient vector is computed:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t,$$

where β_1 is a hyperparameter (smoothing coefficient), $\mathbf{m}_0 = 0$.

3. Estimation of the Hessian diagonal. Every training steps ($t \bmod k = 1$), an estimation of the Hessian diagonal is performed. In [83], two estimators for the Hessian diagonal are proposed: Hutchinson's unbiased estimator [44] and the Gauss-Newton-Bartlett (GNB) estimator [45]. The GNB estimator is only applicable for cross-entropy loss functions in classification tasks and is therefore unsuitable for training PINNs with loss functions like mean squared error (MSE). The universal estimator is Hutchinson's estimator, which uses stochastic approximation:

$$\text{diag}(\mathbf{H}) \approx \mathbf{v} \mathbf{e} \mathbf{H} \mathbf{v},$$

where \mathbf{H} is the Hessian matrix (hessian) of the loss function $L(\boldsymbol{\theta})$, $\boldsymbol{\theta}$ is the network's parameter vector, including weights and RBF parameters, $\mathbf{v}^{(m)}$ are random vectors distributed according to a multivariate normal distribution with zero mean and unit variance or according to a Rademacher distribution where vector components take values ± 1 with equal probability, \mathbf{e} denotes element-wise multiplication.

The product of the unknown Hessian \mathbf{H} and the vector \mathbf{v} is computed without explicitly constructing the Hessian using Pearlmutter's trick [46]:

$$\mathbf{H} \mathbf{v} = \nabla_{\boldsymbol{\theta}} (\mathbf{v}^T \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})).$$

Hutchinson's estimator is applicable for training PINNs but is computationally expensive. Therefore, in this work, as an approximation of the Hessian diagonal, it is proposed to use the vector obtained by the element-wise multiplication of the loss function gradient vector with itself:

$$\hat{\mathbf{h}}_t = \mathbf{g}_t \mathbf{e} \mathbf{g}_t.$$

The estimator accounts for the local curvature of the loss function. In networks, weights and function parameters influence convergence differently. The proposed estimator automatically balances steps for heterogeneous parameters. Furthermore, for many PDEs with high-frequency solutions, the approximate preconditioner smooths noise more strongly, while a more accurate Hessian calculation might amplify instability.

4. Smoothing the Hessian estimate (EMA). The obtained estimate $\hat{\mathbf{h}}_t$ is smoothed using exponential moving average:

$$\mathbf{h}_t = \beta_2 \mathbf{h}_{t-k} + (1 - \beta_2) \hat{\mathbf{h}}_t.$$

5. Update clipping. To ensure stability with inaccurate Hessian estimates and control instability due to the non-convexity of the loss function, a clipping function $\text{clip}(d, s) = \max(\min(d, s), -s)$ is applied:

$$\mathbf{d}_t = \text{clip}\left(\frac{\mathbf{m}_t}{\max(\gamma \cdot \mathbf{h}_t, \varepsilon)}, 1\right),$$

where γ is a control parameter, ε is a small constant.

This step guarantees the algorithm's stability.

6. Parameter update. The network parameters are updated according to the formula:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}'_t - \eta \mathbf{d}_t,$$

where η is the learning rate.

In non-convex regions, Sophia behaves similarly to SignGD (Sign Gradient Descent). SignGD is a simplified variant of gradient descent where instead of the gradient itself, only its sign is used:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}' - \eta \text{sign}(\nabla L(\boldsymbol{\theta}')),$$

where $\text{sign}(\nabla L(\boldsymbol{\theta}'))$ is the element-wise sign function, which returns +1 for positive values, returns -1 for negative values, returns 0 for zero, η is the learning rate.

The SignGD method is invariant to the gradient scale because the update size is always η or $-\eta$. This is useful in problems with non-uniform curvature where gradients across different coordinates differ significantly. SignGD is also robust to noise as it uses only the gradient's direction, not its absolute magnitude. It can be viewed as a limiting case of Adam or RMSProp. However, the convergence time of SignGD depends on the ratio of maximum to minimum

curvature. Sophia improves upon SignGD by introducing an adaptive preconditioner based on a Hessian estimate and update clipping. Instead of a fixed step η , it uses:

$$\mathbf{d}' = \text{clip}\left(\frac{\mathbf{m}'}{\max(\gamma \cdot \mathbf{h}', \epsilon)}, 1\right),$$

where \mathbf{h}' is the curvature estimate.

This accelerates convergence on flat regions of the loss function. Thus, Sophia retains the stability of SignGD but outperforms it due to its second-order adaptability.

2.2 Architectures and Operational Algorithms of PIRBFN for Solving Hydrodynamics Problems

Let's consider three classic test problems for verification and analysis of computational algorithms in hydrodynamics: Kovasznay flow, lid-driven cavity flow, and the Taylor-Green vortex. The first two problems have analytical solutions, making them convenient for testing numerical methods, and the lid-driven cavity problem serves as a standard model for studying the behavior of viscous fluid in closed domains.

Kovasznay flow [47] is a special case of two-dimensional stationary Navier-Stokes equations for an incompressible fluid. For an incompressible medium, the density is considered constant [48]: $\rho = \text{const}$, where ρ — is the fluid density. In the stationary case, partial derivatives with respect to time are zero, giving:

$$\frac{\partial v}{\partial t} = 0, \quad \frac{\partial u}{\partial t} = 0.$$

The system of two-dimensional stationary Navier-Stokes equations in the absence of external forces is written as:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{1}$$

$$u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{\text{Re}} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \tag{2}$$

$$u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{\text{Re}} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \tag{3}$$

where u and v — are the components of the velocity vector along the x and y axes respectively, p — is the pressure, Re — is the Reynolds number.

For Kovasznay flow, an exact analytical solution is known:

$$u(x, y) = -e^{-\lambda x} \cos(2\pi y), \quad v(x, y) = -\frac{\lambda}{2\pi} e^{-\lambda x} \sin(2\pi y), \quad p(x, y) = -\frac{1}{2} e^{-2\lambda x}.$$

where $u(x, y)$ and $v(x, y)$ — are the velocity components, $p(x, y)$ — is the pressure.

The parameter λ is determined by the Reynolds number Re according to the relation [48]

$$\lambda = \sqrt{\frac{\text{Re}^2}{4} + 4\pi^2} - \frac{\text{Re}}{2}.$$

We choose the unit square as the computational domain $\Omega = [-1, 1] \times [-1, 1]$. Boundary conditions follow from the

$$u(x, y) = 1 - e^{-\lambda x} \cos(2\pi y), \quad v(x, y) = \frac{\lambda}{2\pi} e^{-\lambda x} \sin(2\pi y), \quad p(x, y) = -\frac{1}{2} e^{-2\lambda x}, \quad (x, y) \in \partial\Omega.$$

analytical solution:

The loss function when modeling Kovasznay flow with PIRBFN is formulated as the sum of squared residuals of the equations and boundary conditions:

$$I = \frac{1}{N} \sum_{i=1}^N \left(u_i \frac{\partial v_i}{\partial x} + v_i \frac{\partial v_i}{\partial y} + \frac{\partial p_i}{\partial y} - \frac{1}{\text{Re}} \left(\frac{\partial^2 v_i}{\partial x^2} + \frac{\partial^2 v_i}{\partial y^2} \right) \right)^2 + \frac{\lambda}{K} \sum_{j=1}^K (u_j - u_j^A)^2 + \frac{1}{N} \sum_{i=1}^N \left(u_i \frac{\partial v_i}{\partial x} + v_i \frac{\partial v_i}{\partial y} + \frac{\partial p_i}{\partial y} - \frac{1}{\text{Re}} \left(\frac{\partial^2 v_i}{\partial x^2} + \frac{\partial^2 v_i}{\partial y^2} \right) \right)^2 + \frac{\lambda}{K} \sum_{j=1}^K (v_j - v_j^A)^2 + \frac{1}{N} \sum_{i=1}^N \left(u_i \frac{\partial v_i}{\partial x} + v_i \frac{\partial v_i}{\partial y} + \frac{\partial p_i}{\partial y} - \frac{1}{\text{Re}} \left(\frac{\partial^2 v_i}{\partial x^2} + \frac{\partial^2 v_i}{\partial y^2} \right) \right)^2 + \frac{\lambda}{K} \sum_{j=1}^K (v_j - v_j^A)^2 + \tag{4}$$

$$\frac{1}{N} \sum_{i=1}^N \left(\frac{\partial u_i}{\partial x} + \frac{\partial v_i}{\partial y} \right)^2 + \frac{\lambda}{K} \sum_{j=1}^K (p_j - p_j^A)^2.$$

To solve the system (1)–(3) three independent PIRBFNs are employed: two for approximating the velocity components and one for pressure. All networks are united by a common loss function. Next, the gradients of the loss function (4) with respect to the network parameters are found, which are used by various optimization algorithms to update the weights.

The structure of PIRBFN allows for analytically finding the components of the gradient vector of the loss function with respect to the network parameters. However, this approach is labor-intensive, not universal, and prone to errors, leading to cumbersome formulas. Therefore, in practice, it is advisable to apply automatic differentiation methods, which ensure efficient and accurate computation of derivatives directly within the computational graph.

The Taylor-Green vortex represents a transient decaying vortex flow, which serves as an analytical model for studying transitional processes in fluid in the absence of external disturbances [49]. The physical nature of the vortex is related to the evolution of an initial periodic velocity field, which at the initial moment forms a regular structure of interacting vortices. Subsequently, nonlinear redistribution of energy between different spatial scales of the flow occurs.

Unlike many hydrodynamics problems where vorticity is generated due to the no-slip condition at walls, in the Taylor-Green vortex, turbulent processes are isolated from boundary effects. Dissipation and energy transfer are caused solely by viscosity and the nonlinear terms of the Navier-Stokes equations. In the two-dimensional case, thanks to entropy conservation, vortices are not subject to the cascade decay characteristic of three-dimensional turbulence, which allows studying mechanisms of laminar decay.

At the initial moment, a smooth periodic velocity distribution leads to the formation of ordered vortex cells separated by zones of intense deformation – saddle (hyperbolic) points. In vortex centers (for example, at points $(\pi/2, \pi/2)$ and $(3\pi/2, 3\pi/2)$) vorticity is maximal, and velocity is close to zero, corresponding to the rotation of fluid particles. At saddle points (such as points $(\pi/2, 3\pi/2)$ and $(0, \pi)$) velocity and vorticity vanish; intense deformation and mixing of fluid occur here. Over time, viscosity ensures the transfer of vorticity from vortex centers to saddle regions, modeling the process of decay and transition to small-scale structures without the influence of external boundaries.

The two-dimensional Navier-Stokes equations for an incompressible medium have the form:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \quad (5)$$

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} = -\frac{1}{\rho} \frac{\partial p}{\partial x} + \frac{\mu}{\rho} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + F_x, \quad (6)$$

$$\frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} = -\frac{1}{\rho} \frac{\partial p}{\partial y} + \frac{\mu}{\rho} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) + F_y, \quad (7)$$

where u and v – are the velocity components, p – is the pressure, F_x – is an external force (for example, the force of gravity), μ – is the dynamic viscosity, ρ – is the fluid density (in this model problem the density is constant), t – is time.

In the absence of gravity and with unit density, the equations (5)–(7) are written as:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \quad (8)$$

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \mu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right), \quad (9)$$

$$\frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \mu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right). \quad (10)$$

The Taylor-Green vortex has an analytical solution:

$$u(x, y, t) = \sin(x) \cos(y) e^{(-2\nu t)}, \quad v(x, y, t) = -\cos(x) \sin(y) e^{(-2\nu t)}, \quad p(x, y, t) = \frac{1}{4} (\cos(2x) + \cos(2y)) e^{(-4\nu t)},$$

where ν — is the kinematic viscosity, $u(x, y, t)$ and $v(x, y, t)$ — are the velocity components, $p(x, y, t)$ — is the pressure, t — is time.

The initial conditions are written as:

$$u(x, y, 0) = \sin(x)\cos(y)e^{(-2\nu t)}, \quad v(x, y, 0) = -\cos(x)\sin(y)e^{(-2\nu t)}, \quad p(x, y, 0) = \frac{1}{4}(\cos(2x) + \cos(2y)).$$

The initial conditions satisfy the continuity equation (8) and the momentum equations (9) and (10) for the velocity field of an incompressible fluid.

Typically, the Taylor-Green vortex is considered in a periodic square solution domain $[0, 2\pi] \times [0, 2\pi]$. Based on this, the boundary conditions can be represented as follows:

$$\begin{aligned} u(0, y, t) &= u(2\pi, y, t), & u(x, 0, t) &= u(x, 2\pi, t), & v(0, y, t) &= v(2\pi, y, t), \\ v(x, 0, t) &= v(x, 2\pi, t), & p(0, y, t) &= p(2\pi, y, t), & p(x, 0, t) &= p(x, 2\pi, t). \end{aligned}$$

Thus, pressure and velocity components are periodic across the boundaries. The flow is confined to a square domain. Due to the absence of wall influence, this model problem is convenient for spectral methods.

The solution of the system (8)–(10) is performed using three PIRBFNs: two networks for determining the two components of the velocity vector and one network for determining the pressure. The problem has three independent variables, so the coordinate vectors of the centers have three components, distributed in the cubic domain $[0, 2\pi] \times [0, 2\pi] \times [0, 1]$. The three networks are linked by a common error functional:

$$\begin{aligned} I &= \frac{1}{N} \sum_{i=1}^N \left(\frac{\partial u_i}{\partial t} + u_i \frac{\partial u_i}{\partial x} + v_i \frac{\partial u_i}{\partial y} + \frac{\partial p_i}{\partial x} - \mu \left(\frac{\partial^2 u_i}{\partial x^2} + \frac{\partial^2 u_i}{\partial y^2} \right) \right)^2 + \frac{\lambda}{K} \sum_{j=1}^K (u_j - u_j^A)^2 + \\ &\frac{1}{N} \sum_{i=1}^N \left(\frac{\partial v_i}{\partial t} + u_i \frac{\partial v_i}{\partial x} + v_i \frac{\partial v_i}{\partial y} + \frac{\partial p_i}{\partial y} - \mu \left(\frac{\partial^2 v_i}{\partial x^2} + \frac{\partial^2 v_i}{\partial y^2} \right) \right)^2 + \frac{\lambda}{K} \sum_{j=1}^K (v_j - v_j^A)^2 + \\ &\frac{1}{N} \sum_{i=1}^N \left(\frac{\partial u_i}{\partial x} + \frac{\partial v_i}{\partial y} \right)^2 + \frac{\lambda}{K} \sum_{j=1}^K (p_j - p_j^A)^2. \end{aligned} \quad (11)$$

The gradients of the error functional are sequentially found (11). The found gradients are applied by various algorithms to update all trainable network parameters.

Lid-driven cavity flow in a square cavity [50, 51] is a classic two-dimensional stationary test case for incompressible Navier-Stokes equations. This model is widely used in computational fluid dynamics to analyze the interaction of inertial and viscous forces in a closed domain. The geometric simplicity of the problem (a square with side L) combines with rich vortex dynamics arising solely due to the motion of the top boundary.

The top wall (lid) moves with a constant horizontal velocity U , creating shear stress and setting the initially stationary fluid in motion. The flow pattern is entirely determined by the dimensionless Reynolds number:

$$\text{Re} = (U \cdot L) / \nu,$$

where L — is the side length of the cavity, ν — is the kinematic viscosity. This parameter reflects the ratio between inertial forces, which promote vortex formation, and viscous forces, which suppress motion.

At low Reynolds numbers, viscosity dominates, and the flow remains laminar, with a smooth shear profile, without pronounced vortex structures. With an increase in the Reynolds number (approximately in the range $10 \leq \text{Re} \leq 1000$), inertial effects intensify, leading to the formation of a large primary vortex in the center of the cavity, rotating in the direction of the lid's motion. In corners, especially lower ones, secondary vortices with opposite rotation appear, caused by flow separation. With further increase in Re , tertiary vortices of smaller scale may appear, complicating the flow pattern. At Reynolds numbers exceeding a critical value (on the order of several thousand), the flow loses its stationarity and becomes unsteady, demonstrating a transition to a turbulent regime.

The model is widely used for testing numerical methods, as well as for modeling mixing processes, flows in closed tanks, and ventilated cavities.

Within the problem, an incompressible fluid with constant density is considered [48]. For stationary flow, time derivatives are zero, so the two-dimensional Navier-Stokes equations have the form:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \quad (12)$$

$$u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{\text{Re}} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right), \quad (13)$$

$$u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{\text{Re}} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right), \quad (14)$$

where u and v — are the horizontal and vertical velocity components, p — is the pressure, Re — is the Reynolds number.

Boundary conditions for velocity are set by no-slip (Dirichlet) conditions:

For the top wall ($y = L$): $u(x, L) = U, \mathfrak{V}(x, L) = 0$.

For the bottom wall ($y = 0$): $u(x, 0) = 0, \mathfrak{V}(x, 0) = 0$.

For the left wall ($x = 0$): $u(0, y) = 0, \mathfrak{V}(0, y) = 0$.

For the right wall ($x = L$): $u(L, y) = 0, \mathfrak{V}(L, y) = 0$.

For pressure, Neumann conditions are formulated, following from the projection of the momentum equation onto the normal to the wall. On all boundaries, the relation holds:

$$\frac{\partial p}{\partial n} = 0.$$

where n — is the unit normal vector to the corresponding boundary.

The solution of the system (12)–(14) is performed using three PIRBFNs: two networks approximate the velocity components, the third — the pressure field. All networks are linked by a common error functional, the minimization of which ensures satisfaction of the equations and boundary conditions:

$$\begin{aligned} I = & \frac{1}{N} \sum_{i=1}^N \left(u_i \frac{\partial u_i}{\partial x} + v_i \frac{\partial u_i}{\partial y} + \frac{\partial p_i}{\partial x} - \frac{1}{\text{Re}} \left(\frac{\partial^2 u_i}{\partial x^2} + \frac{\partial^2 u_i}{\partial y^2} \right) \right)^2 + \frac{\lambda}{K} \sum_{j=1}^K (u_j - u_j^A)^2 + \\ & \frac{1}{N} \sum_{i=1}^N \left(u_i \frac{\partial v_i}{\partial x} + v_i \frac{\partial v_i}{\partial y} + \frac{\partial p_i}{\partial y} - \frac{1}{\text{Re}} \left(\frac{\partial^2 v_i}{\partial x^2} + \frac{\partial^2 v_i}{\partial y^2} \right) \right)^2 + \frac{\lambda}{K} \sum_{j=1}^K (v_j - v_j^A)^2 + \\ & \frac{1}{N} \sum_{i=1}^N \left(\frac{\partial u_i}{\partial x} + \frac{\partial v_i}{\partial y} \right)^2 + \frac{\lambda}{K} \sum_{j=1}^K (p_j - p_j^A)^2. \end{aligned} \quad (15)$$

Here, a multiplier is introduced to simplify gradient computation. Training of the networks is carried out by iterative updating of parameters based on the gradients of the functional (15).

3. Results

In Python, using the Keras and TensorFlow libraries, custom extensions were implemented, including a Gaussian radial basis function layer, modified training algorithms for PIRBFN with adjustment of weights and RBF parameters (gradient descent, Nesterov, ADAM, Sophia, Levenberg-Marquardt algorithms), as well as PIRBFN loss functions. All experiments were conducted in the free version of Google Colab using only the CPU. Training of each network continued until a loss function value of was reached 10^{-4} .

Modeling Kovasznay Flow. To solve this problem, three PIRBFNs were implemented. Each network used 64 RBFs, whose centers were initially placed on a uniform square grid 8×8 . RBF parameters and weights were initialized with random values close to zero. During the experiments, the optimal number of neurons and trial points was selected for each network. Inside the solution domain, 100 random collocation points were generated, and on the boundary — 40.

Figures 1–6 present the obtained solutions, corresponding analytical solutions, and modeling errors. Visually, the solutions obtained using PIRBFN demonstrate high similarity to the analytical ones. Errors for all variables remain small: for the first velocity component — in the range from 0 to 0.045, for the second — from 0 to 0.027, for pressure — from 0 to 0.032.

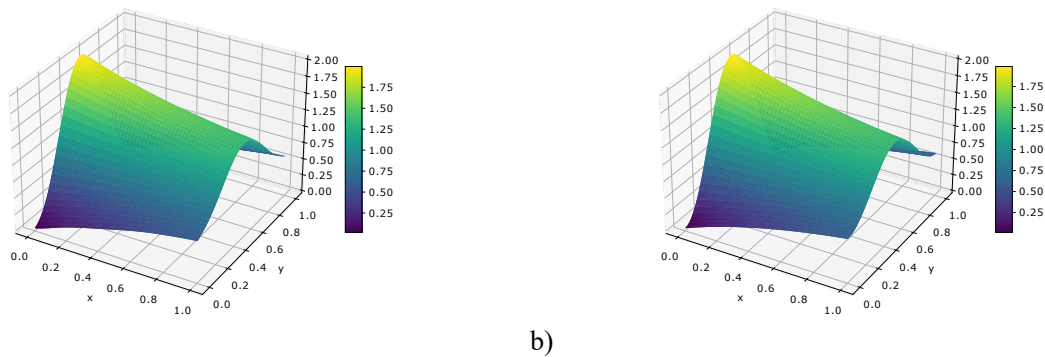


Figure 1. Comparison of the x-velocity component (u) for Kovaszny flow ($Re=40$): (a) PIRBFN-predicted solution, (b) analytical solution. The visual similarity demonstrates the model's high accuracy.

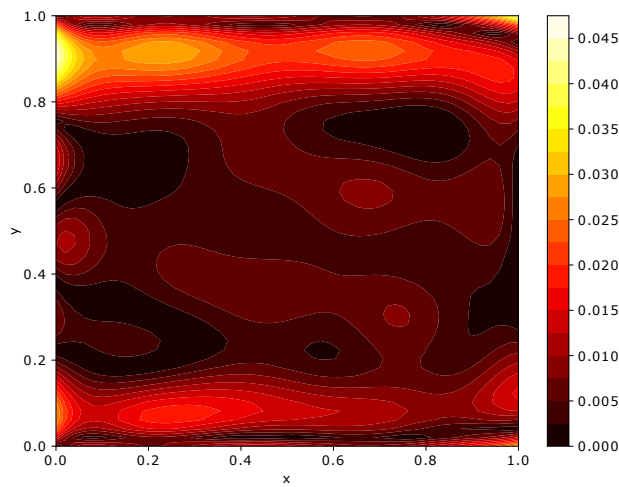


Figure 2. Error of the x-component of the velocity (u) for the Kovashnaya flow ($Re=40$). The difference between the solution obtained using PIRBFN and the analytical solution is quite small.

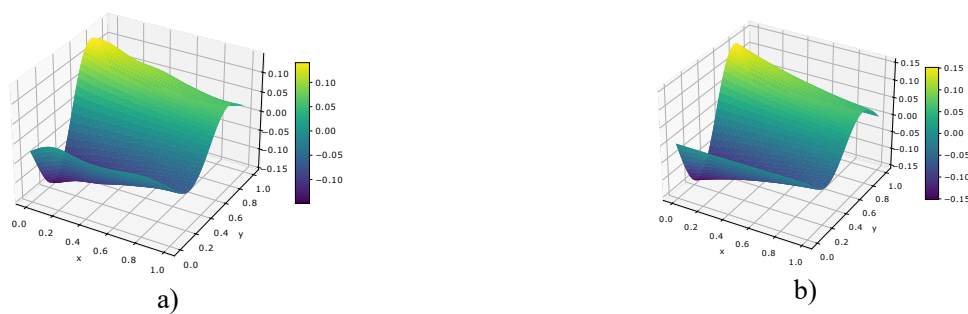


Figure 3. Comparison of the y-component of the velocity (u) for the Kovashnaya flow ($Re=40$).a) a solution predicted by PIRBFN b) analytical solution.

The visual similarity demonstrates the model's high accuracy

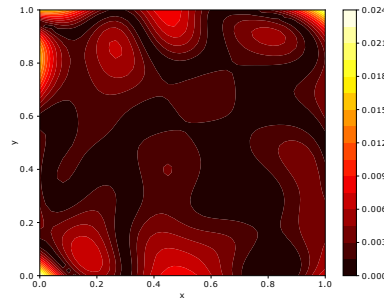


Figure 4. Error of the y-component of the velocity (u) for the Kovashnaya flow ($Re=40$). The difference between the solution obtained using PIRBFN and the analytical solution is quite small.

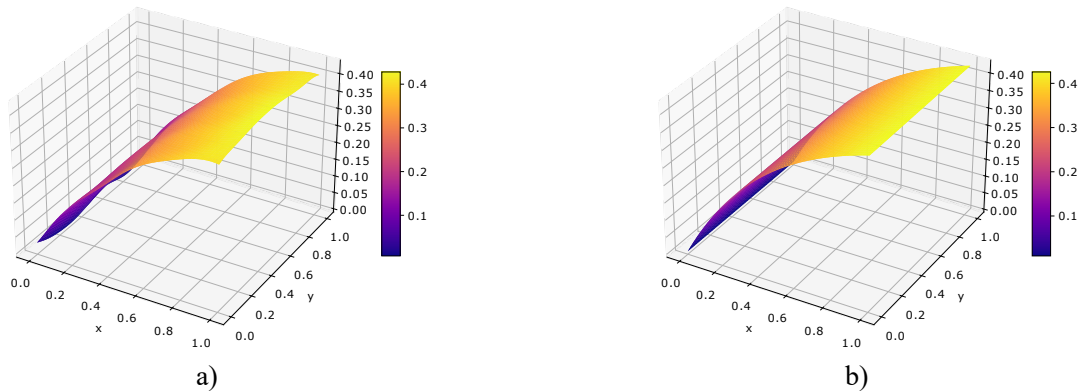


Figure 5. Comparison of the pressure for the Kovashnaya flow ($Re=40$). a) a solution predicted by PIRBFN b) analytical solution.

The visual similarity demonstrates the model's high accuracy

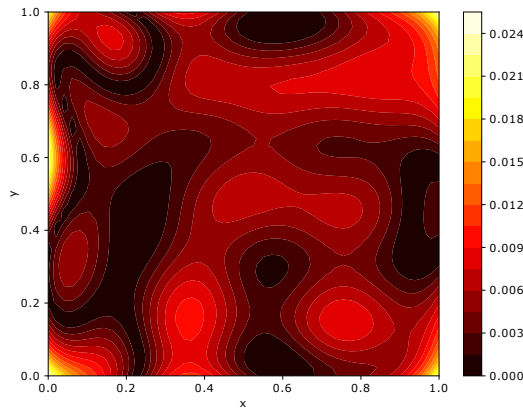


Figure 6. Error of the pressure for the Kovashnaya flow ($Re=40$). The difference between the solution obtained using PIRBFN and the analytical solution is quite small.

Modeling the Taylor-Green Vortex. For this three-dimensional problem, three PIRBFNs were also implemented. Each radial basis function is characterized by a three-component center vector, uniformly distributed in the cubic domain $[0, 2\pi] \times [0, 2\pi] \times [0, 1]$. The networks used 512 RBFs with initial center placement in the cubic domain $8 \times 8 \times 8$. Network parameters were initialized with values close to zero. For each network, the optimal number of neurons and trial points was selected, using 100 collocation points inside the domain and 40 on the boundary.

Figures 7–12 show the solutions at time $t = 0,5$. Visual comparison with the analytical solution shows good agreement. Modeling errors also remain within small limits: for the first velocity component — up to 0.045, for the second — up to 0.027, for pressure — up to 0.032.

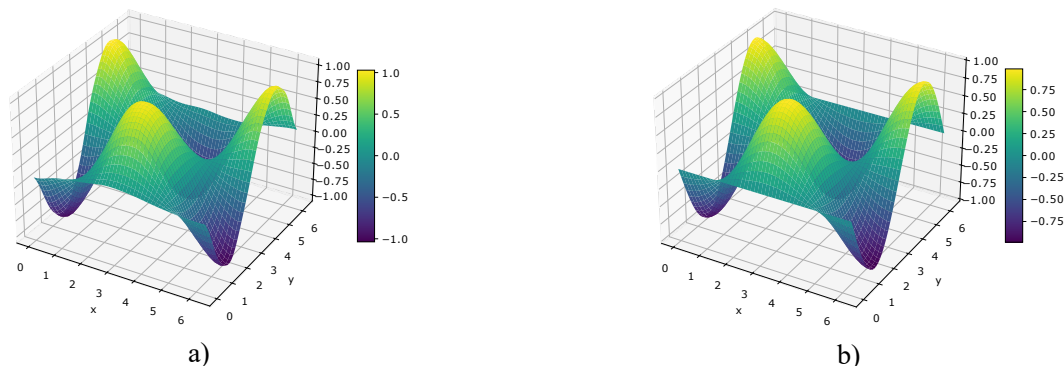


Figure 7. Comparison of the x-component of the velocity (u) for the Taylor-Green Vortex. a) a solution predicted by PIRBFN b) analytical solution.

The visual similarity demonstrates the model's high accuracy

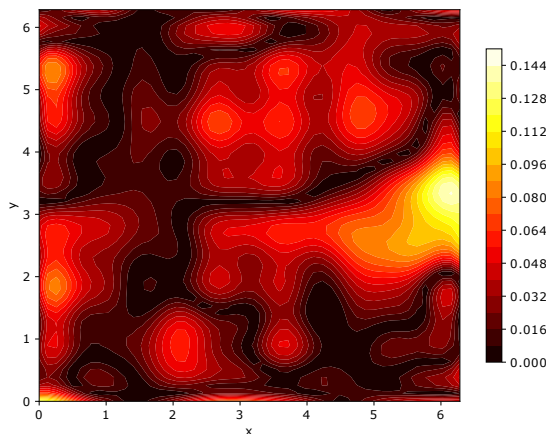


Figure 8. Error of the x-component of the velocity (u) for the Taylor-Green Vortex). The difference between the solution obtained using PIRBFN and the analytical solution is quite small.

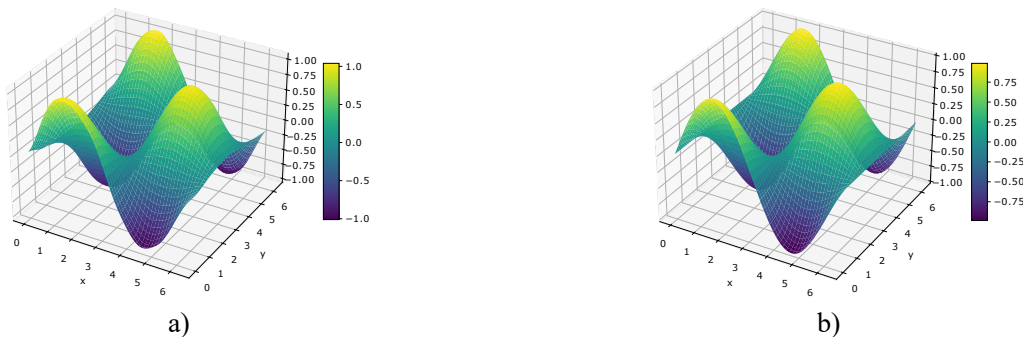


Figure 9. Comparison of the y-component of the velocity (u) for the Taylor-Green Vortex a) a solution predicted by PIRBFN b) analytical solution.

The visual similarity demonstrates the model's high accuracy

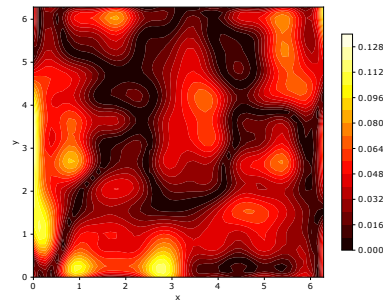


Figure 10. Error of the y-component of the velocity (u) for the Taylor-Green Vortex). The difference between the solution obtained using PIRBFN and the analytical solution is quite small.



Figure 11. Comparison of the pressure for the Taylor-Green Vortex. a) a solution predicted by PIRBFN b) analytical solution

The visual similarity demonstrates the model's high accuracy

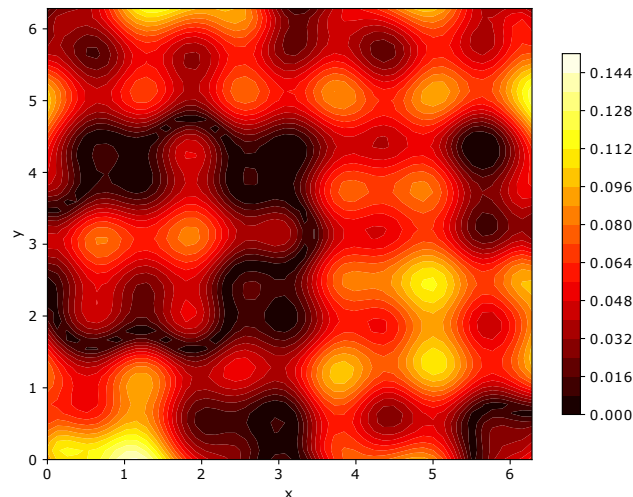


Figure 12. Error of the pressure for the Taylor-Green Vortex). The difference between the solution obtained using PIRBFN and the analytical solution is quite small.

Modeling Lid-Driven Cavity Flow. For modeling the lid-driven cavity flow, three PIRBFNs were constructed. Each network consisted of 64 neurons, whose centers were initially placed on a square grid 8×8 . RBF parameters and weights were initialized with values close to zero. During the tuning process, the optimal number of neurons and trial points was chosen, with 100 collocation points used inside the domain and 40 on the boundary.

The solution for the first velocity component is shown in Fig. 13, the solution for the second velocity component is shown in Fig. 14.

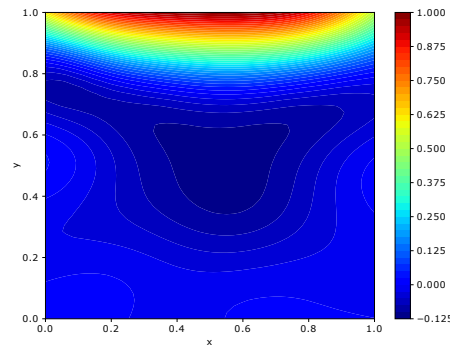


Figure 13. Solution for the x-component of the velocity (u) for the Lid-Driven Cavity Flow. A physically accurate picture of the formation of the x-component has been demonstrated. At the top surface, $u \approx 1$; on the remaining walls, $u = 0$. A vortex with negative velocity is observed at the center.

Figure 13 demonstrates the distribution of the first velocity component u . A profile characteristic of cavity flow is observed, with maximum values in the central area and a decrease in velocity near the walls.

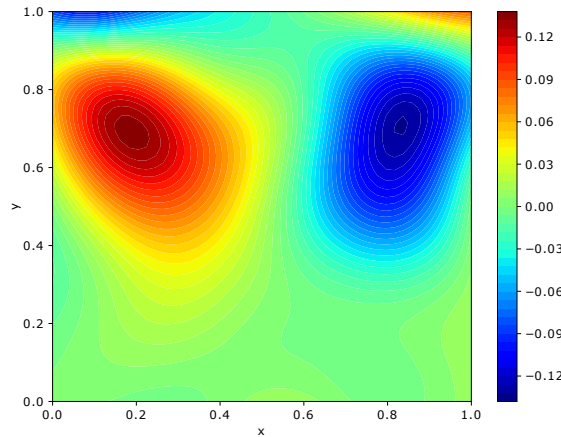


Figure 14. Solution for the y-component of the velocity (v) for the Lid-Driven Cavity Flow. The pattern of y-component formation corresponds to the physics of the problem. The values at the boundary are zero or close to zero; the graph demonstrates antisymmetry: on the left, the flow rises upward; on the right, the flow descends downward.

The pressure solution is shown in Fig. 15.

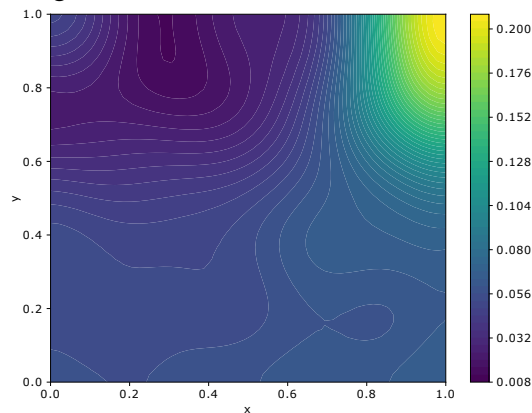


Figure 15. Pressure for the Lid-Driven Cavity Flow. The pressure distribution agrees with hydrodynamic patterns: increased pressure in flow stagnation zones near the walls.

The velocity vector field is shown in Fig. 16.

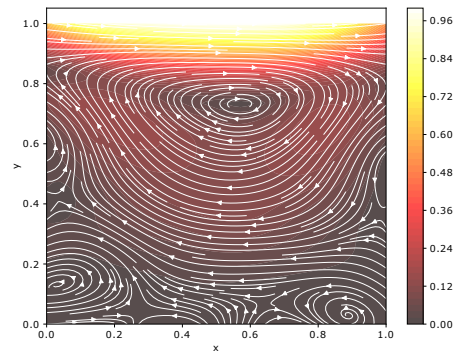


Figure 16. Velocity vector field for flow in a cavity under the action of a moving lid. A physically correct picture of the formation of circulation zones is demonstrated, including secondary vortices in the corners of the cavity, which is typical for this type of flow.

Visual consistency with known physical phenomena and the smoothness of the fields without discontinuities indicate that the model is working correctly.

4. Discussion

In this work, a comparison of the efficiency of solving three model problems in hydrodynamics was performed using various modifications of training algorithms for physics-informed radial basis function networks (PIRBFN), in which not only linear weights but also the parameters of the RBF basis functions are adjusted. The tasks of modeling Kovasznay flow, the Taylor-Green vortex, and lid-driven cavity flow were considered. All computational experiments were conducted in the Google Colab environment under identical conditions and continued until reaching the same specified mean squared error value 10^{-4} .

For comparison, networks were also trained using the popular Adam algorithm with adjustment of only the weights. Quantitative results of the experiments are presented in the table.

Table 1. Comparison of Different PIRBFN Training Algorithms

Training Algorithm	Task					
	Kovasznay Flow		Taylor-Green Vortex		Lid-Driven Cavity Flow	
	Number of Iterations	Solution Time, S	Number of Iterations	Solution Time, S	Number of Iterations	Solution Time, S
Adam Algorithm (training weights only)	12800	1950	14300	2310	13200	2170
Gradient Descent Algorithm	8240	1170	9340	1290	8530	1205
Nesterov Algorithm	5800	810	6200	815	5900	750
Adam Algorithm (adjusted)	4255	700	5910	805	5590	745
Sophia Algorithm (gradient square diagonal Hessian)	1540	105	1870	125	1480	95
Sophia Algorithm (Hutchinson's estimator)	1730	135	2000	140	1590	130
Levenberg-Marquardt Algorithm	310	75	570	95	450	70

An objective indicator of the computational complexity of training PIRBFN is the number of optimization iterations. However, this indicator does not account for differences in the time costs of performing a single iteration. The absolute solution time depends on the hardware and software environment, but under fixed conditions allows for correct relative comparison of training algorithms.

The obtained results clearly confirm the fundamental importance of adjusting RBF parameters during PIRBFN training. Even using the simplest gradient descent algorithm with simultaneous optimization of weights and RBF parameters allowed for a reduction in the number of iterations by more than 1.5 times compared to traditional training

of only weights. A further reduction in the number of iterations is achieved by using the developed modified first and second-order algorithms.

The greatest effect is demonstrated by the second-order algorithm — the modified Levenberg-Marquardt algorithm. It reduces the number of problem-solving iterations compared to the traditional Adam algorithm by a factor ranging from 25 to 41.3, and the total solution time — by a factor from 24.3 to 31.0. For each of the considered tasks, the solution time did not exceed 2 minutes, whereas the traditional Adam algorithm required more than 30 minutes of computation. The modified Sophia algorithm with gradient-square diagonal Hessian approximation (gradient square diagonal Hessian) reduces the number of iterations by approximately 8 times compared to the traditional Adam algorithm and, thanks to the absence of the need to solve a system of linear algebraic equations at each iteration, reduces the solution computation time by a factor of 19–24, which is comparable to the results of the Levenberg–Marquardt algorithm. The Sophia Algorithm (gradient-square-diagonal Hessian) demonstrated a significant reduction in the number of iterations and solution time compared to the modified Sophia algorithm, which computes the Hessian diagonal using the Hutchinson approximation.

Let's compare the PINN approach with traditional grid-based numerical methods for solving partial differential equations. Constructing computational grids for complex curvilinear domains is a non-trivial and resource-intensive task. PINNs implement a mesh-free approach and do not require explicit grid construction. In grid methods, it is typically necessary to solve high-dimensional systems of linear algebraic equations, often with poor conditioning. Moreover, an accurate error estimate of the solution is difficult and usually limited to an approximation order estimate. The error in solving an ill-conditioned system of linear equations can significantly exceed the residual norm.

In the PINN approach, the number of stages where numerical errors accumulate is smaller, and the training result directly gives the value of the residual of the original partial differential equation. Additionally, grid methods are oriented primarily towards solving direct problems and require full specification of boundary conditions. PINNs, on the contrary, use a unified formalism for direct and inverse problems, allow working with arbitrary points in space, are effective for complex and changing geometry, as well as in the presence of noisy or sparse data. The main disadvantage of classical PINNs remains significant training time. However, the results obtained for PIRBFN show that the gap in solution time between neural network and traditional numerical methods can be significantly reduced.

Let's consider the differences between PINNs built on fully-connected neural networks and PIRBFN. In PIRBFN, the approximation of the target function is local: each basis function describes the solution in the vicinity of its center. This provides a fundamental advantage compared to fully-connected PINNs because parameter gradients become localized, and significant contributions to the loss function are formed only in limited regions of space. Such a structure allows for effective reduction of local PDE residuals.

In classical PINNs, only the linear weights of the network are optimized. In PIRBFN, the nonlinear parameters of the basis functions are additionally adjusted, allowing the optimizer to adapt the shape of the basis functions (through width vectors) and their position (through centers). This is especially important for correctly describing sharp gradients and local features of the solution. In fully-connected PINNs, on the contrary, due to the global compositional structure of the network, local features require coordinated retuning of a large number of parameters across all layers, which significantly slows down training [52].

Since PIRBFNs contain one nonlinear layer followed by a linear output layer, the loss function surface is generally less complex than that of deep neural networks. Shallow architectures with localized bases are characterized by a shorter effective length of gradient propagation paths and a smaller number of interacting nonlinearities between layers. This substantially simplifies the optimization process, especially in PINNs, where the physical loss function often leads to conflicts between gradients of its different components.

Results presented in works on PIRBFN confirm that RBF networks with trainable basis parameters converge to PDE solutions in a significantly smaller number of iterations compared to standard PINNs, including problems with high-frequency components and ill-conditioned domains. For example, for modeling Kovasznay flow in this work, an order of magnitude fewer iterations were required than in [7]. For the lid-driven cavity flow problem, the number of iterations was approximately 7 times less than in the work [53].

Thus, adjusting RBF parameters expands the effective hypothesis space in a structured manner, consistent with the characteristic features of partial differential equation solutions. This leads to a radical reduction in the number of effective gradient optimization steps required to minimize the physics-informed loss function, even when using the same optimizer. Precisely for this reason, PIRBFNs are trained substantially faster than fully-connected PINNs in many tasks.

5. Conclusion

Physics-informed neural networks (PINNs) have established themselves as an effective tool for solving boundary value problems described by partial differential equations (PDEs). Despite their conceptual universality and flexibility, their widespread practical application is still limited by two key factors: the high computational cost of training and limited efficiency when solving more complex problem classes.

Physics-informed radial basis function networks (PIRBFNs) represent a promising direction for overcoming these limitations. Due to their structural features, PIRBFNs typically require fewer training iterations compared to traditional fully-connected PINNs, especially when jointly adjusting network weights and radial basis function parameters. This additional adaptability contributes to accelerated convergence and improved numerical stability.

One of the most significant application areas for PIRBFNs is solving the Navier-Stokes equations, which form the basis for the mathematical description of hydrodynamics. In this work, modified PIRBFN training algorithms were developed, including ADAM, Sophia, and Levenberg-Marquardt algorithms, which provide for the joint optimization of weights and basis function parameters. These algorithms enhance both the speed and flexibility of the training process.

The proposed PIRBFN-based implementations were tested on benchmark computational fluid dynamics problems, including Kovasznay flow, the Taylor-Green vortex, and lid-driven cavity flow. In all three cases, the modified PIRBFN training algorithms demonstrated a reduction in computational time while maintaining high solution accuracy. The conducted numerical experiments confirm the high potential of PIRBFNs for modeling fluid mechanics problems, including situations where analytical solutions are unavailable.

The results on these benchmark problems suggest that PIRBFNs with joint parameter optimization are a promising approach for solving the incompressible Navier-Stokes equations. The dramatic reduction in training time compared to fully-connected PINNs makes them a more practical tool for problems where computational cost is a barrier. However, further investigation is needed to assess their scalability to fully 3D, time-dependent turbulent flows, where the number of required basis functions and the complexity of the loss landscape may pose new challenges.

Acknowledgements

This study was financially supported by the Ministry of Science and Higher Education of the Russian Federation within the framework of the project "Physics-Informed Neural Networks for Modeling Distributed Parameter Systems" (Registration Number 126021217095-4).

References

- [1] Lagaris IE, Likas A, Fotiadis DI. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans Neural Netw.* 1998;9(5):987-1000. doi:10.1109/72.712178.
- [2] Yadav N, Yadav A, Kumar M. An introduction to neural network methods for differential equations. Dordrecht: Springer; 2015. doi:10.1007/978-94-017-9816-7.
- [3] Raissi M, Perdikaris P, Karniadakis GE. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. *arXiv.* 2017. Preprint. doi:10.48550/arXiv.1711.10561.
- [4] Raissi M, Perdikaris P, Karniadakis GE. Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations. *arXiv.* 2017. Preprint. doi:10.48550/arXiv.1711.10566.
- [5] Raissi M, Perdikaris P, Karniadakis GE. 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. doi:10.1016/j.jcp.2018.10.045.
- [6] Lakshmanan V, Robinson S, Munn M. *Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps.* 1st ed. Sebastopol: O'Reilly Media; 2020.
- [7] Jin X, Cai S, Li H, Karniadakis GE. NSFnets (Navier-Stokes Flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *J Comput Phys.* 2021;426:109951. doi:10.1016/j.jcp.2020.109951.
- [8] Awerweg S, Schwarz A, Schwarz C, Schröder J. 3D modeling of generalized Newtonian fluid flow with data assimilation using the least-squares finite element method. *Comput Methods Appl Mech Eng.* 2022;392:114668. doi:10.1016/j.cma.2022.114668.
- [9] Zhang L-C, Lee D. Design-based individual prediction. *arXiv.* 2023. Preprint. doi:10.48550/arXiv.2301.09117.

- [10] Karniadakis GE, Kevrekidis IG, Lu L, Perdikaris P, Wang S, Yang L. Physics-informed machine learning. *Nat Rev Phys*. 2021;3:422-440. doi:10.1038/s42254-021-00314-5.
- [11] Zubov K, McCarthy Z, Ma Y, et al. NeuralPDE: Automating Physics-Informed Neural Networks (PINNs) with Error Approximations. arXiv. 2021. Preprint. doi:10.48550/arXiv.2107.09443.
- [12] Taha WM, Taha AE, Thunberg J. *Cyber-Physical Systems: A Model-Based Approach*. Cham: Springer; 2021. doi:10.1007/978-3-030-36071-9.
- [13] Lu Y, Liu C, Wang K, Huang H, Xu X. Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues. *Robot Comput Integr Manuf*. 2020;61:101837. doi:10.1016/j.rcim.2019.101837.
- [14] Uhlemann THJ, Steinhilper CLR, Steinhilper R. The Digital Twin: Realizing the Cyber-Physical Production System for Industry 4.0. *Procedia CIRP*. 2017;61:335-340. doi:10.1016/j.procir.2016.11.152.
- [15] Tarkhov DA, Malykhina GF. Neural network modelling methods for creating digital twins of real objects. *J Phys Conf Ser*. 2019;1236:012068. doi:10.1088/1742-6596/1236/1/012068.
- [16] Tarkhov DA, Vasilyev AN. *Semi-Empirical Neural Network Modeling and Digital Twins Development*. Cambridge: Academic Press; 2019.
- [17] Raissi M, Perdikaris P, Karniadakis GE. 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. doi:10.1016/j.jcp.2018.10.045.
- [18] Lu L, Meng X, Mao Z, Karniadakis GE. DeepXDE: A deep learning library for solving differential equations. *SIAM Rev*. 2021;63(1):208-228. doi:10.1137/19M1274067.
- [19] Cai S, Wang Z, Wang S, Perdikaris P, Karniadakis GE. Physics-Informed Neural Networks (PINNs) for Heat Transfer. *J Heat Transfer*. 2021;143(6):060801. doi:10.1115/1.4050542.
- [20] Geron A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 3rd ed. Sebastopol: O'Reilly Media; 2022.
- [21] Bavdin AG, Pearlmutter BF, Radul AA, Siskind JM. Automatic Differentiation in Machine Learning: a Survey. *J Mach Learn Res*. 2018;18(153):1-43. doi:10.48550/arXiv.1502.05767.
- [22] Strizhak SV, Koshelev KB. The use of a physically based neural network on the example of modeling hydrodynamic processes that allow an analytical solution. *Proc Inst Syst Program RAS*. 2023;35(5):245-258. doi:10.15514/ISPRAS-2023-35(5)-16.
- [23] Sophiya AA, Nair AK, Maleki S, Krishnababu SK. A comprehensive analysis of PINNs: Variants, Applications, and Challenges. arXiv. 2025. Preprint. doi:10.48550/arXiv.2505.22761.
- [24] Gorbachenko VI, Stenkin DA. Solving Equations Describing Processes in a Piecewise Homogeneous Medium on Radial Basis Functions Networks. In: *Studies in Computational Intelligence*. Vol 925. Cham: Springer; 2021:412-419. doi:10.1007/978-3-030-60577-3_49.
- [25] Gorbachenko VI, Stenkin DA. Solving of Inverse Coefficient Problems on Networks of Radial Basis Functions. In: *Studies in Computational Intelligence*. Vol 1008. Cham: Springer; 2022:230-237. doi:10.1007/978-3-030-91581-0_31.
- [26] Cybenko G. Approximation by Superposition of a Sigmoidal Function. *Math Control Signals Syst*. 1989;2:303-314. doi:10.1007/BF02134016.
- [27] Maiorov V, Pinkus A. Lower bounds for approximation by MLP neural networks. *Neurocomputing*. 1999;25(1-3):81-91. doi:10.1016/S0925-2312(98)00111-8.
- [28] Hornik K. Approximation capabilities of multilayer feedforward networks. *Neural Netw*. 1991;4(2):251-257. doi:10.1016/0893-6080(91)90009-T.
- [29] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. *Neural Netw*. 1989;2(5):359-366. doi:10.1016/0893-6080(89)90020-8.
- [30] Leshno M, Lin VY, Pinkus A, Schocken S. Multilayer Feedforward Networks with a Non-Polynomial Activation Function Can Approximate Any Function. *Neural Netw*. 1993;6(6):861-867. doi:10.1016/S0893-6080(05)80131-5.
- [31] Osowski S. *Sieci neuronowe do przetwarzania informacji*. Warszawa: Oficyna Wydawnicza Politechniki Warszawskiej; 2000.
- [32] Haykin S. *Neural Networks and Learning Machines*. 3rd ed. London: Pearson; 2009.
- [33] Park J, Sandberg IW. Approximation and Radial-Basis-Function Networks. *Neural Comput*. 1993;5(2):305-316. doi:10.1162/neco.1993.5.2.305.
- [34] Park J, Sandberg IW. Universal Approximation Using Radial-Basis-Function Networks. *Neural Comput*. 1991;3(2):246-257. doi:10.1162/neco.1991.3.2.246.

- [35] Buhmann MD. *Radial Basis Functions: Theory and Implementations*. Cambridge: Cambridge University Press; 2003.
- [36] Micchelli CA. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constr Approx*. 1986;2(1):11-22. doi:10.1007/BF01893414.
- [37] Eivazi H, Tahani M, Schlatter P, Vinuesa R. Physics-informed neural networks for solving Reynolds-averaged Navier–Stokes equations. *Phys Fluids*. 2022;34(7):075117. doi:10.1063/5.0095270.
- [38] De Ryck T, Jagtap AD, Mishra S. Error estimates for physics-informed neural networks approximating the Navier–Stokes equations. *IMA J Numer Anal*. 2024;44(1):83-119. doi:10.1093/imanum/drac085.
- [39] Demirkaya G, Wafo SC, Ilegbusi OJ. Direct solution of Navier–Stokes equations by radial basis functions. *Appl Math Model*. 2008;32(9):1848-1858. doi:10.1016/j.apm.2007.06.019.
- [40] Chinchapatnam PP, Djidjeli K, Nair PB, Tan M. A compact RBF-FD based meshless method for the incompressible Navier–Stokes equations. *Proc Inst Mech Eng M: J Eng Marit Environ*. 2009;223(3):275-290. doi:10.1243/14750902JEME151.
- [41] Abdukhakimov F, Xiang C, Kamzolov D, Gower R, Takáč M. SANIA: Polyak-type Optimization Framework Leads to Scale Invariant Stochastic Algorithms. arXiv. 2023. Preprint. doi:10.48550/arXiv.2312.17369.
- [42] Nagwekar A. Towards Guided Descent: Optimization Algorithms for Training Neural Networks At Scale. arXiv. 2024. Preprint. doi:10.48550/arXiv.2512.18373.
- [43] Liu H, Li Z, Hall D, Liang P, Ma T. Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training. arXiv. 2023. Preprint. doi:10.48550/arXiv.2305.14342.
- [44] Hutchinson MF. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Commun Stat Simul Comput*. 1989;18(3):1059-1076. doi:10.1080/03610919008812866.
- [45] Schraudolph NN. Fast curvature matrix-vector products for second-order gradient descent. *Neural Comput*. 2002;14(7):1723-1738. doi:10.1162/08997660260028683.
- [46] Pearlmutter BA. Fast Exact Multiplication by the Hessian. *Neural Comput*. 1994;6(1):147-160. doi:10.1162/neco.1994.6.1.147.
- [47] Kovasznay LSG. Laminar flow behind a two-dimensional grid. *Math Proc Camb Philos Soc*. 1948;44(1):58-62. doi:10.1017/S0305004100023999.
- [48] Slezkin NA. *Dynamics of viscous incompressible fluid*. Moscow: Gostekhizdat; 1955. Russian.
- [49] Taylor GI, Green AE. Mechanism of the Production of Small Eddies from Large Ones. *Proc R Soc Lond A*. 1937;158(895):499-521. doi:10.1098/rspa.1937.0036.
- [50] Botella O, Peyret R. Benchmark spectral results on the lid-driven cavity flow. *Comput Fluids*. 1998;27(4):421-433. doi:10.1016/S0045-7930(98)00002-4.
- [51] Ghia U, Ghia KN, Shin CT. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *J Comput Phys*. 1982;48(3):387-411. doi:10.1016/0021-9991(82)90058-4.
- [52] Xie T, Yu H, Wilamowski BM. Comparison between Traditional Neural Networks and Radial Basis Function Networks. In: 2011 IEEE International Symposium on Industrial Electronics; 2011 Jun 27-30; Gdansk, Poland. IEEE; 2011. p. 1194-1199. doi:10.1109/ISIE.2011.5984334.
- [53] Wang S, Teng Y, Perdikaris P. Understanding and mitigating gradient pathologies in physics-informed neural networks. arXiv. 2020. Preprint. doi:10.48550/arXiv.2001.04536.