ISSN 0027-1349, Moscow University Physics Bulletin, 2024, Vol. 79, Suppl. 2, pp. S914–S921. © Allerton Press, Inc., 2024.

Improving Physics-Informed Neural Networks via Quasiclassical Loss Functionals

S. G. Shorokhov*

RUDN University, Moscow, Russia

Received October 1, 2024; revised October 10, 2024; accepted October 20, 2024

Abstract—We develop loss functionals for training physics–informed neural networks using variational principles for nonpotential operators. Generally, a quasiclassical variational functional is bounded from above or below, contains derivatives of lower order compared to the order of derivatives in partial differential equation and some boundary conditions are integrated into the functional, which results in lower computational costs when evaluating the functional via Monte Carlo integration. Quasiclassical loss functional of boundary value problem for hyperbolic equation is obtained using the symmetrizing operator by V.M. Shalov. We demonstrate convergence of the neural network training and advantages of quasiclassical loss functional over conventional residual loss functional of boundary value problems for hyperbolic equation.

Keywords: physics-informed neural networks, partial differential equations, variational principle, loss functional

DOI: 10.3103/S0027134924702370

1. INTRODUCTION

Physics-informed neural networks (PINNs) [1] are an innovative approach to the investigation of partial differential equations (PDEs). PINNs incorporate the knowledge of governing PDEs and their boundary conditions as well as any physical laws relevant to the physical problem under consideration.

Basically, PDEs and boundary conditions are injected into PINNs via the neural network loss (error) functional [2], using the residual loss for the governing PDE and the residual losses for each of the boundary (initial, terminal) conditions of the boundary value problem (BVP).

Neural networks for the approximation of BVP solutions may be trained with residual loss functionals [1, 3] or energy loss functionals [4]. Some researchers [5, 6] suggest, in the absence of classical (energy) functionals, to use nonclassical functionals arising from variational principles for nonpotential operators [7] for neural network training.

We study if a variational functional from [7] can be used as a loss functional for training PINN that approximate the solution to a BVP for hyperbolic PDE.

*E-mail: shorokhov-sg@rudn.ru

2. BOUNDARY VALUE PROBLEM FOR HYPERBOLIC EQUATION

One of the most promising approaches to variational principles for nonpotential operators is the method of symmetrizing operator by Shalov [8, 9]. This method implies that the BVP (PDE and boundary conditions) under study is represented in the vector form

$$\mathbf{A}\,u = \mathbf{f}, \tag{1}$$

where $\mathbf{A}$ is a vector operator in a Hilbert space, $\mathbf{f}$ is a vector function, u is an unknown solution to the BVP.

If one finds a vector symmetrizing operator $\mathbf{B}$ such that operator $\mathbf{A}$ is $\mathbf{B}$-symmetric:

$$\langle \mathbf{A}u,\, \mathbf{B}v\rangle = \langle \mathbf{B}u,\, \mathbf{A}v\rangle \;\forall u, v$$

(the notation $\langle *,\, *\rangle$ stands for the scalar product) and $\mathbf{B}$-positive:

$$\langle \mathbf{A}u,\, \mathbf{B}u\rangle > 0 \,\forall u \neq 0,$$

$$\langle \mathbf{A}u_n,\, \mathbf{B}u_n\rangle \underset{n\to\infty}{\longrightarrow} 0 \Rightarrow \|u_n\| \underset{n\to\infty}{\longrightarrow} 0,$$

then the variational functional for the BVP (1) can be obtained in the form [8, 9]

$$D\,[u] = \langle \mathbf{A}\,u,\, \mathbf{B}\,u\rangle - 2\,\langle \mathbf{f},\, \mathbf{B}\,u\rangle\,. \tag{2}$$

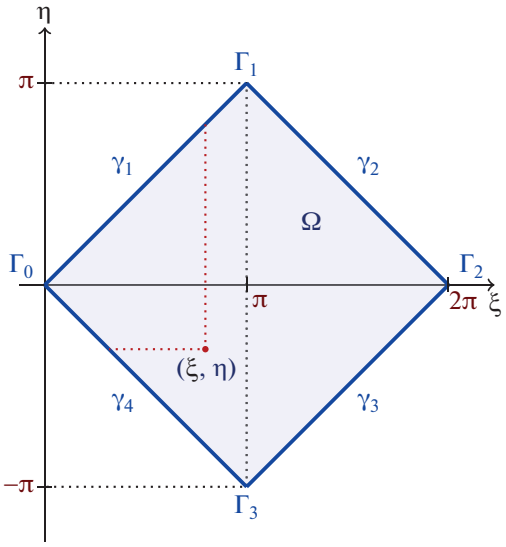

Fig. 1. Boundary value problem domain Ω.

The method of symmetrizing operator was successfully applied to BVPs for hyperbolic PDE [10, 11], parabolic (heat) PDE [12, 13], hypoelliptic PDE [14] and other PDEs [15, 16].

We study variational principle for a basic homogeneous hyperbolic equation

$$u_{\xi\eta} = 0 \tag{3}$$

in a rhombus-shaped domain Ω with vertices at the points $\Gamma_0(0,\,0)$, $\Gamma_1(\pi,\,\pi)$, $\Gamma_2(2\pi,\,0)$, $\Gamma_3(\pi,\,-\pi)$ (see Fig. 1). The boundary $\partial\Omega$ of domain Ω consists of four segments $\gamma_1, \gamma_2, \gamma_3, \gamma_4$, where the segment γ_1 connects the points Γ_0 and Γ_1, the segment γ_2 connects the points Γ_1 and Γ_2, the segment γ_3 connects the points Γ_2 and Γ_3, the segment γ_4 connects the points Γ_3 and Γ_0.

Boundary conditions for PDE (3) are given in the form

$$\begin{cases} u\,|_{\gamma_1} = & \chi_1(\xi), \\ u_\eta\,|_{\gamma_2} = & \varphi_2(\eta), \\ u_\eta\,|_{\gamma_3} = & \varphi_3(\eta), \\ u_\xi\,|_{\gamma_3} = & \psi_3(\xi), \\ u_\xi\,|_{\gamma_4} = & \psi_4(\xi), \end{cases} \tag{4}$$

where the functions on the right-hand side of (4) are L_2-functions and the boundary conditions (4) are consistent. In (3), (4) and below, a greek letter subscript denotes partial derivative with respect to the corresponding variable.

The solution u of BVP (3)–(4) belongs to $W_2^1(\Omega)$ and $(\chi_1, \varphi_2, \varphi_3, \psi_3, \psi_4) \in L_2(\gamma_1 \times \gamma_2 \times \gamma_3 \times \gamma_3 \times \gamma_4)$, where L_2 is the space of square-integrable functions, W_2^1 is the Sobolev space of all locally square-integrable functions that admit (weak) first-order derivative.

The components of the outer normal $\vec{n}$ to the boundary $\partial\Omega$ on different sections of the boundary $\gamma_1, \gamma_2, \gamma_3, \gamma_4$ are obviously calculated as follows:

$$\vec{n}(\gamma_1) = \frac{1}{\sqrt{2}}(-1,\,1),$$
$$\vec{n}(\gamma_2) = \frac{1}{\sqrt{2}}(1,\,1),$$
$$\vec{n}(\gamma_3) = \frac{1}{\sqrt{2}}(1,\,-1),$$
$$\vec{n}(\gamma_4) = \frac{1}{\sqrt{2}}(-1,\,-1).$$

The vector differential operator $\mathbf{A}$ and the vector function $\mathbf{f}$ in operator equation (1) are determined from (3) and (4) as follows:

$$\mathbf{A}\,u = \begin{bmatrix} u_{\xi\eta} \\ u \\ u_\eta \\ u_\eta \\ u_\xi \\ u_\xi \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} 0 \\ \chi_1 \\ \varphi_2 \\ \varphi_3 \\ \psi_3 \\ \psi_4 \end{bmatrix}.$$

The vector integro-differential operator $\mathbf{B}$, acting from $W_2^1(\Omega)$ to $L_2(\Omega \times \gamma_1 \times \gamma_2 \times \gamma_3 \times \gamma_3 \times \gamma_4)$, can be defined by the equality [10]

$$\mathbf{B}\,v = \begin{bmatrix} \int\limits_{\xi}^{\gamma_1 \cup \gamma_4} v_\eta(\zeta,\eta)\,d\zeta + \int\limits_{\eta}^{\gamma_1 \cup \gamma_2} v_\xi(\xi,\tau)\,d\tau \\ v \\ -n_1(\gamma_2) \int\limits_{\xi}^{\gamma_1} v_\eta(\zeta,\eta)\,d\zeta \\ -n_1(\gamma_3) \int\limits_{\xi}^{\gamma_4} v_\eta(\zeta,\eta)\,d\zeta \\ -n_2(\gamma_3) \int\limits_{\eta}^{\gamma_2} v_\xi(\xi,\tau)\,d\tau \\ -n_2(\gamma_4) \int\limits_{\eta}^{\gamma_1} v_\xi(\xi,\tau)\,d\tau \end{bmatrix}.$$

Vector functions $\mathbf{A}\,u$, $\mathbf{B}\,v$, and $\mathbf{f}$ are defined on the Cartesian product of domains $\Omega \times \gamma_1 \times \gamma_2 \times \gamma_3 \times \gamma_3 \times \gamma_4$.

The scalar product of vectors $\mathbf{A}u$ and $\mathbf{B}v$ is calculated by multiplying the corresponding components,

calculating the integrals over the corresponding sets, and summing the resulting values using the formula

$$\langle \mathbf{A}u,\, \mathbf{B}v\rangle = \int\limits_{\Omega} (\mathbf{A}\,u)_1\,(\mathbf{B}\,v)_1\; d\xi d\eta + \int\limits_{\gamma_1} (\mathbf{A}\,u)_2\,(\mathbf{B}\,v)_2\; ds + \int\limits_{\gamma_2} (\mathbf{A}\,u)_3\,(\mathbf{B}\,v)_3\; ds + \int\limits_{\gamma_3} (\mathbf{A}\,u)_4\,(\mathbf{B}\,v)_4\; ds + \int\limits_{\gamma_3} (\mathbf{A}\,u)_5\,(\mathbf{B}\,v)_5\; ds + \int\limits_{\gamma_4} (\mathbf{A}\,u)_6\,(\mathbf{B}\,v)_6\; ds.$$

Application of multiple integration by parts [17] leads to the formula [10]

$$\langle \mathbf{A}u,\, \mathbf{B}v\rangle = \int\limits_{\Omega} (u_\xi v_\xi + u_\eta v_\eta)\; d\xi d\eta + \int\limits_{\gamma_1} u\,v\; ds,$$

that ensures $\mathbf{B}$-symmetry and $\mathbf{B}$-positivity of operator $\mathbf{A}$.

Thus, the quasiclassical (bounded from below [10]) variational functional (2) for BVP (3)–(4) is equal to the following sum of integrals

$$D\,[u] = \int\limits_{\Omega} \left(u_\xi^2 + u_\eta^2\right)\; d\xi d\eta + \int\limits_{\gamma_1} u^2\; ds - 2\int\limits_{\gamma_1} \chi_1 u\; ds + 2\int\limits_{\gamma_2} \varphi_2 n_1 \int\limits_{\xi}^{\gamma_1} u_\eta\,(\zeta,\eta)\; d\zeta\; ds + 2\int\limits_{\gamma_3} \varphi_3 n_1 \sin\xi \int\limits_{\xi}^{\gamma_4} u_\eta\,(\zeta,\eta)\; d\zeta\; ds + 2\int\limits_{\gamma_3} \psi_3 n_2 \int\limits_{\eta}^{\gamma_2} u_\xi\,(\xi,\tau)\; d\tau\; ds + 2\int\limits_{\gamma_4} \psi_4 n_2 \int\limits_{\eta}^{\gamma_1} u_\xi\,(\xi,\tau)\; d\tau\; ds, \qquad (5)$$

where n_1, n_2 denote the components of the normal vector $\vec{n}$ on the sections $\gamma_1, \gamma_2, \gamma_3, \gamma_4$ of the boundary $\partial\Omega$.

Monte Carlo methods [18] provide a reliable and efficient approach to estimation of integral losses in deep learning, because Monte Carlo sampling allows to approximate integrals at reduced cost and prevents gradient descent convergence to a local minimum. However, functional (5) is hard to evaluate with Monte Carlo integration [18] when training a neural network due to necessity of Monte Carlo sampling on each level of each repeated (iterated) integral in (5).

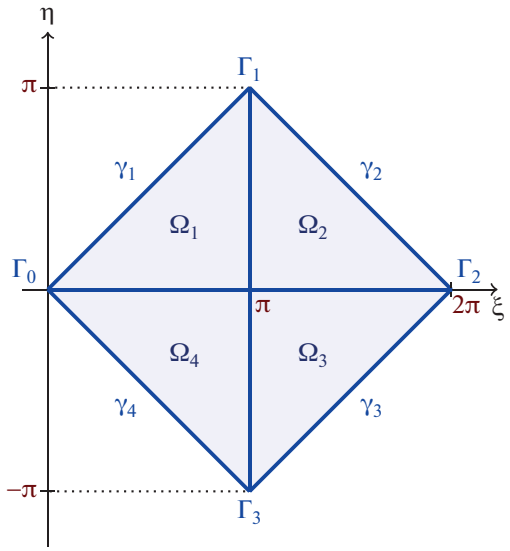

Fig. 2. Subdomains Ω_1, Ω_2, Ω_3, and Ω_4 of domain Ω.

The domain Ω is equal to the union of four subdomains $\Omega_1, \Omega_2, \Omega_3, \Omega_4$ (see Fig. 2).

It can be proven that the repeated integrals in (5) can be represented as integrals over subdomains of the domain Ω:

$$\int\limits_{\gamma_2} \varphi_2\,(\eta)\; n_1 \int\limits_{\xi}^{\gamma_1} u_\eta\,(\zeta,\eta)\; d\zeta\; ds = -\int\limits_{\Omega_1\cup\Omega_2} \varphi_2\,(\eta)\; u_\eta\,(\xi,\eta)\; d\xi d\eta,$$

$$\int\limits_{\gamma_3} \varphi_3\,(\eta)\; n_1 \sin\xi \int\limits_{\xi}^{\gamma_4} u_\eta\,(\zeta,\eta)\; d\zeta\; ds = -\int\limits_{\Omega_3\cup\Omega_4} \varphi_3\,(\eta)\; u_\eta\,(\xi,\eta)\; d\xi d\eta,$$

$$\int\limits_{\gamma_3} \psi_3\,(\xi)\; n_2 \int\limits_{\eta}^{\gamma_2} u_\xi\,(\xi,\tau)\; d\tau\; ds = -\int\limits_{\Omega_2\cup\Omega_3} \psi_3\,(\xi)\; u_\xi\,(\xi,\eta)\; d\xi d\eta,$$

$$\int\limits_{\gamma_4} \psi_4\,(\xi)\; n_2 \int\limits_{\eta}^{\gamma_1} u_\xi\,(\xi,\tau)\; d\tau\; ds = -\int\limits_{\Omega_1\cup\Omega_4} \psi_4\,(\xi)\; u_\xi\,(\xi,\eta)\; d\xi d\eta.$$

We define auxiliary functions $\Phi(\eta), -\pi \leqslant \eta \leqslant \pi$ and $\Psi(\xi), 0 \leqslant \xi \leqslant 2\pi$ as follows:

$$\Phi(\eta) = \begin{cases} \varphi_2(\eta), & 0 \leqslant \eta \leqslant \pi, \\ \varphi_3(\eta), & -\pi \leqslant \eta < 0, \end{cases}$$

$$\Psi(\xi) = \begin{cases} \psi_3(\xi), & \pi \leqslant \xi \leqslant 2\pi, \\ \psi_4(\xi), & 0 \leqslant \xi < \pi. \end{cases}$$

Now, we can transform functional (5) to the following compact form.

The variational functional (5) for the BVP (3), (4) can be represented as

$$D[u] = \int\limits_{\Omega} \left(u_\xi^2 + u_\eta^2 - 2\Phi u_\eta - 2\Psi u_\xi\right) d\xi d\eta + \int\limits_{\gamma_1} \left(u^2 - 2\chi_1 u\right) ds. \quad (6)$$

3. NEURAL NETWORK MODEL OF BOUNDARY VALUE PROBLEM

The solution $u(\xi, \eta)$ of hyperbolic PDE (3) can be approximated by a deep neural network with the output $f(\xi, \eta; \boldsymbol{\theta})$, where ξ, η are the input values of the neural network, and $\boldsymbol{\theta}$ is the vector of the neural network parameters (weights and biases):

$$u(\xi, \eta) \approx f(\xi, \eta; \boldsymbol{\theta}).$$

We will build and train the neural network for the BVP in the domain Ω (Fig. 1) with PDE (3) and boundary conditions

$$\begin{cases} u|_{\gamma_1} = \frac{1}{10\pi}\xi, \\ u_\eta|_{\gamma_2} = -\frac{1}{2}\sin 2\eta + \frac{1}{10\pi}, \\ u_\eta|_{\gamma_3} = -\frac{1}{2}\sin 2\eta + \frac{1}{10\pi}, \\ u_\xi|_{\gamma_3} = \frac{1}{2}\sin 2\xi, \\ u_\xi|_{\gamma_4} = \frac{1}{2}\sin 2\xi. \end{cases} \quad (7)$$

The BVP (3), (7) admits an exact solution

$$u = \frac{1}{2}\sin(\xi + \eta)\sin(\xi - \eta) + \frac{1}{10\pi}\eta, \quad (8)$$

which is displayed in Fig. 3.

Computational experiments are carried out for a feedforward neural network (FF) with dense layers with the following hyperparameters:

– the number of hidden layers is 4 with the activation function tanh (hyperbolic tangent),

– the number of neurons in the hidden layers is 100,

– one neuron without activation function is in the output layer,

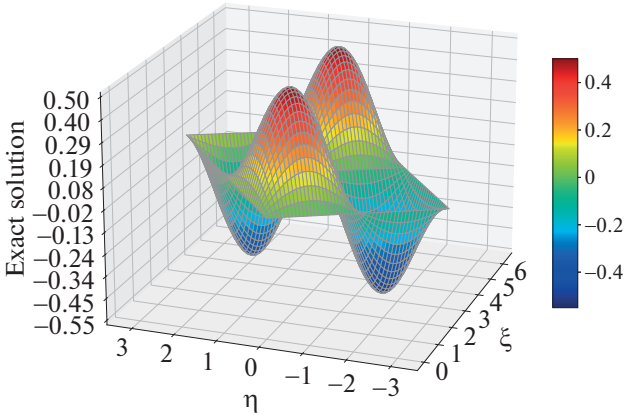

Fig. 3. 3D surface of exact solution (8) to BVP (3), (7).

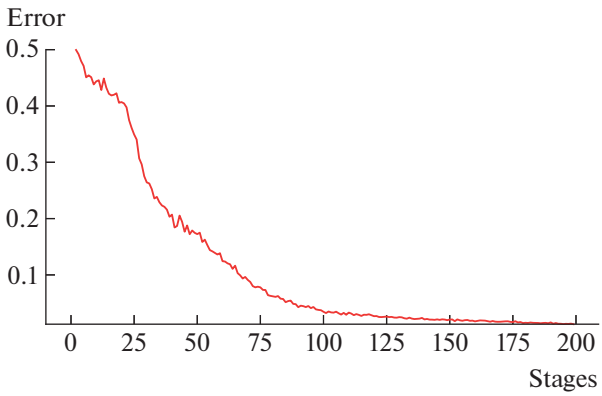

Fig. 4. The learning curve of PINN training with residual loss functional (10).

– 200 training epochs (stages) are performed with 10 stochastic gradient descent (SGD) steps for each epoch (stage),

– the Adam optimizer [19] is used with an initial learning rate of 0.0001,

– a batch contains 1000 samples from domain Ω and 500 samples for each segment of the boundary $\partial\Omega$.

We use the residual functional and the constructed quasiclassical functional (6) as loss functionals during neural network training.

Computer experiments are performed on Apple M2 Max chip with 12-core CPU and 38-core GPU. The program code for building and training the neural networks is implemented using TensorFlow framework [20].

The performance of trained PINNs will be evaluated with a sample $\{(\xi_i, \eta_i)\}_{i=1}^n$ of n points from Ω using mean squared error (MSE), mean absolute error (MAE) and coefficient of determination (R^2):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(u(\xi_i, \eta_i) - f(\xi_i, \eta_i; \boldsymbol{\theta})\right)^2,$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|u\left(\xi_i,\eta_i\right) - f\left(\xi_i,\eta_i;\boldsymbol{\theta}\right)\right|,$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(u\left(\xi_i,\eta_i\right) - f\left(\xi_i,\eta_i;\boldsymbol{\theta}\right)\right)^2}{\sum_{i=1}^{n}\left(u\left(\xi_i,\eta_i\right) - \bar{f}\right)^2}, \quad (9)$$

where $\bar{f} = \frac{1}{n}\sum_{i=1}^{n} f\left(\xi_i,\eta_i;\boldsymbol{\theta}\right)$.

4. TRAINING NEURAL NETWORK WITH RESIDUAL LOSS FUNCTIONAL

The residual loss functional of PINN for BVP (3), (4) is defined by the formula

$$\mathcal{L}_R\left(f\right) = \int\limits_{\Omega} f_{\xi\eta}^2\, d\xi d\eta + \int\limits_{\gamma_1}\left(f - \chi_1\right)^2 ds + \int\limits_{\gamma_2}\left(f_\eta - \varphi_2\right)^2 ds + \int\limits_{\gamma_3}\left(f_\eta - \varphi_3\right)^2 ds + \int\limits_{\gamma_3}\left(f_\xi - \psi_3\right)^2 ds + \int\limits_{\gamma_4}\left(f_\xi - \psi_4\right)^2 ds. \quad (10)$$

The main steps of training PINN for BVP (3), (4) with residual loss functional (10) and Monte Carlo sampling are summarized in Algorithm 1.

When using the residual functional $\mathcal{L}_R\left(f\right)$, one has to evaluate six integrals in (10) and sample from five domains Ω, γ_1, γ_2, γ_3, γ_4, and calculate partial derivatives of the unknown function f up to the second order (see Algorithm 1).

The learning curve of training PINN for BVP (3), (7) with residual loss functional (10) is shown in Fig. 4. It takes 125.7 s to complete 200 stages of PINN training.

The heat map of the absolute error of the neural network approximation

$$e_a\left(\xi,\eta\right) = \left|u\left(\xi,\eta\right) - f\left(\xi,\eta;\boldsymbol{\theta}\right)\right|, \quad (\xi,\eta)\in\Omega$$

is displayed in Fig. 5.

The heat map of the relative error of the neural network approximation

$$e_r\left(\xi,\eta\right) = \left|1 - \frac{f\left(\xi,\eta;\boldsymbol{\theta}\right)}{u\left(\xi,\eta\right)}\right|, \quad (\xi,\eta)\in\Omega$$

is displayed in Fig. 6.

Performance metrics (9) of the obtained PINN model of BVP (3), (7) are as follows:

$$MSE = 0.0261, \quad MAE = 0.1074, \quad R^2 = 0.5785.$$

Absolute error of PINN approximation to BVP (3), (7) with residual loss functional (10) is visualized in Fig. 7.

From Fig. 7 it is evident that the quality of PINN approximation deteriorates when $\xi > \pi$.

Algorithm 1. Training PINN for BVP (3), (4) with residual loss functional (10)

Data: boundary functions χ_1, φ_2, φ_3, ψ_3, ψ_4, probability distributions ν_0, ν_1, ν_2, ν_3, ν_4, initial learning rate α_0, absolute tolerance $\epsilon > 0$

Result: optimal PINN parameter set $\boldsymbol{\theta}$;

choose random initial parameter set $\boldsymbol{\theta}_0$;

repeat/* `stochastic gradient descent step` */

Generate five random samples from the domain Ω and the boundary segments γ_1, γ_2, γ_3, γ_4:
– a random sample $\{(\xi_{i_0},\,\eta_{i_0})\}_{i_0=1}^{n_0}$ of n_0 points from Ω with distribution ν_0;
– a random sample $\{(\xi_{i_1},\,\eta_{i_1})\}_{i_1=1}^{n_1}$ of n_1 points from $\gamma_1\subset\partial\Omega$ with distribution ν_1;
– a random sample $\{(\xi_{i_2},\,\eta_{i_2})\}_{i_2=1}^{n_2}$ of n_2 points from $\gamma_2\subset\partial\Omega$ with distribution ν_2;
– a random sample $\{(\xi_{i_3},\,\eta_{i_3})\}_{i_3=1}^{n_3}$ of n_3 points from $\gamma_3\subset\partial\Omega$ with distribution ν_3;
– a random sample $\{(\xi_{i_4},\,\eta_{i_4})\}_{i_4=1}^{n_4}$ of n_4 points from $\gamma_4\subset\partial\Omega$ with distribution ν_4.
Calculate the residual functional $\mathcal{L}_R\left(f\right)$ for the generated random samples combined into a minibatch $\mathbf{s}_k = \{\{(\xi_{i_0},\eta_{i_0})\}_{i_0=1}^{n_0}, \{(\xi_{i_1},\eta_{i_1})\}_{i_1=1}^{n_1}, \{(\xi_{i_2},\eta_{i_2})\}_{i_2=1}^{n_2}, \{(\xi_{i_3},\eta_{i_3})\}_{i_3=1}^{n_3}, \{(\xi_{i_4},\eta_{i_4})\}_{i_4=1}^{n_4}\}$:

$$\mathcal{L}_R\left(f,\mathbf{s}_k;\boldsymbol{\theta}_k\right) \leftarrow \frac{1}{n_0}\sum_{i_0=1}^{n_0}\left(f_{\xi\eta}\left(\xi_{i_0},\,\eta_{i_0};\boldsymbol{\theta}_k\right)\right)^2 + \frac{1}{n_1}\sum_{i_1=1}^{n_1}\left(f\left(\xi_{i_1},\,\eta_{i_1};\boldsymbol{\theta}_k\right) - \chi_1\left(\xi_{i_1}\right)\right)^2 + \frac{1}{n_2}\sum_{i_2=1}^{n_2}\left(f_\eta\left(\xi_{i_2},\,\eta_{i_2};\boldsymbol{\theta}_k\right) - \varphi_2\left(\eta_{i_2}\right)\right)^2 + \frac{1}{n_3}\sum_{i_3=1}^{n_3}\Big(\left(f_\eta\left(\xi_{i_3},\,\eta_{i_3};\boldsymbol{\theta}_k\right) - \varphi_3\left(\eta_{i_3}\right)\right)^2 + \left(f_\xi\left(\xi_{i_3},\,\eta_{i_3};\boldsymbol{\theta}_k\right) - \psi_3\left(\xi_{i_3}\right)\right)^2\Big) + \frac{1}{n_4}\sum_{i_4=1}^{n_4}\left(f_\xi\left(\xi_{i_4},\,\eta_{i_4};\boldsymbol{\theta}_k\right) - \psi_4\left(\xi_{i_4}\right)\right)^2$$

Perform a number of gradient descent steps with a minibatch (random points) $\mathbf{s}_k$ using the adaptive Adam algorithm (or other neural network learning algorithm) with learning rate α_k (the learning rate α_k is updated automatically at each step):

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha_k\nabla_{\boldsymbol{\theta}}\mathcal{L}_R\left(f,\,\mathbf{s}_k;\,\boldsymbol{\theta}_k\right).$$

until $||\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k|| < \epsilon$; `absolute tolerance` */

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{k+1}$

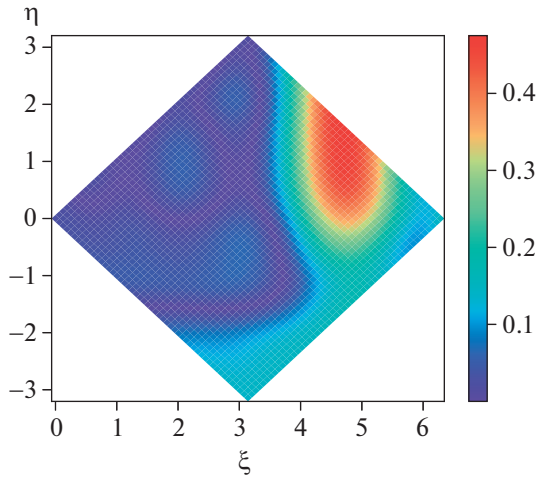

Fig. 5. Absolute error $e_a(\xi, \eta)$ of PINN approximation with residual loss functional (10).

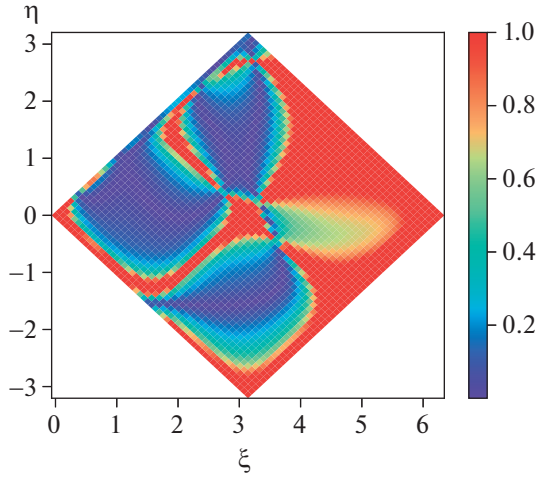

Fig. 6. Relative error $e_r(\xi, \eta)$ of PINN approximation with residual loss functional (10).

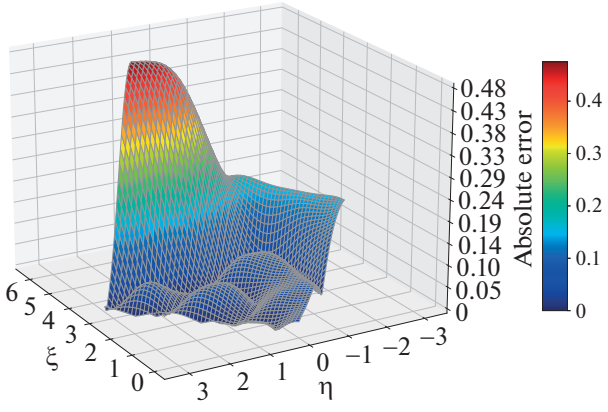

Fig. 7. 3D surface of absolute error of PINN approximation to BVP (3), (7) with residual loss functional (10).

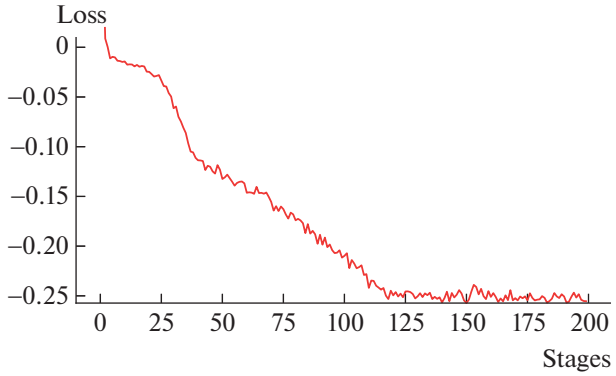

Fig. 8. The learning curve of PINN training with quasiclassical loss functional (11).

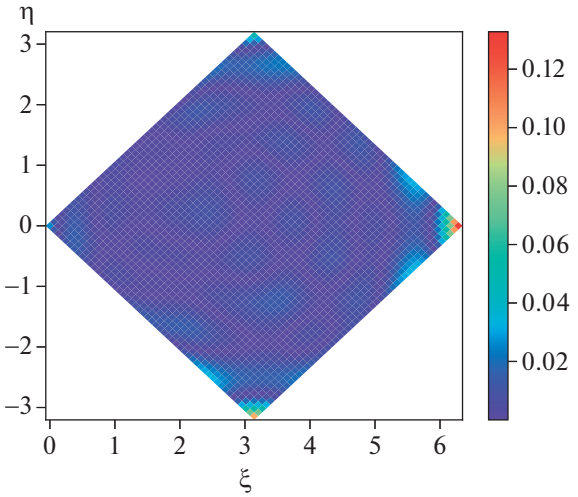

Fig. 9. Absolute error e_a of PINN approximation with quasiclassical loss functional (11).

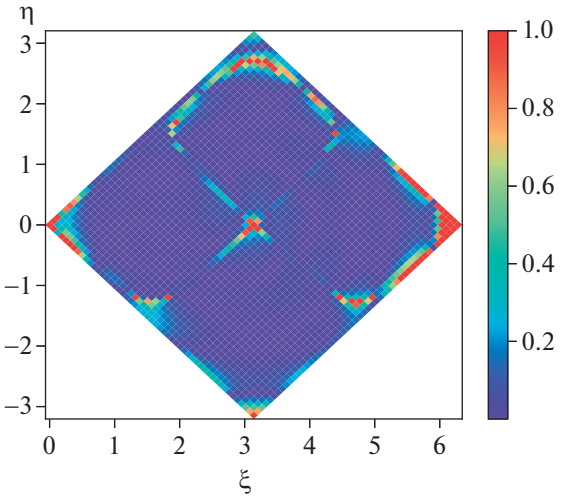

Fig. 10. Relative error e_r of PINN approximation with quasiclassical loss functional (11).

5. TRAINING NEURAL NETWORK WITH QUASICLASSICAL LOSS FUNCTIONAL

Quasiclassical functional (6) can be used to obtain quasiclassical loss functional

$$\mathcal{L}_Q(f) = \int\limits_{\Omega} \left(f_\xi^2 + f_\eta^2 - 2\Phi\, f_\eta - 2\Psi\, f_\xi\right) d\xi d\eta + \int\limits_{\gamma_1} \left(f^2 - 2\chi_1 f\right) ds \tag{11}$$

to train PINN for BVP (3), (4).

The main steps of training PINN for BVP (3), (4) with quasiclassical loss functional (11) and Monte Carlo sampling are summarized in Algorithm 2.

When using the quasiclassical functional $\mathcal{L}_Q(f)$, one has to evaluate two integrals in (11), and, accordingly, to sample from the domain Ω and from the segment γ_1, and calculate first-order partial derivatives of the unknown function f.

Therefore, generally, when using the quasiclassical loss functional (11), training of a neural network requires less computational resources compared to using the residual loss functional (10) due to the use of fewer number of random samples and calculation of partial derivatives of lower order.

The learning curve of training PINN for BVP (3), (7) with quasiclassical loss functional (11) is demonstrated in Fig. 8. It takes 42.7 s to complete 200 stages of PINN training, which is 3 times faster compared to training with residual loss functional (10).

The heat map of the absolute error of the neural network approximation $e_a(\xi, \eta)$ is shown in Fig. 9.

The heat map of the relative error of the neural network approximation $e_r(\xi, \eta)$ is shown in Fig. 10.

Performance metrics (9) of the obtained PINN model of BVP (3), (7) are as follows:

$$MSE = 0.0001, \quad MAE = 0.0061, \quad R^2 = 0.9984,$$

which is substantially better than performance metrics of PINN model, trained with residual loss functional.

Absolute error of PINN approximation to BVP (3), (7) with quasiclassical loss functional (11) is visualized in Fig. 11.

From Fig. 11 it is evident that the quality of PINN model, trained with quasiclassical loss functional (11), is essentially higher than the quality of PINN model, trained with residual loss functional (10), and PINN approximation with loss functional (11) deteriorates only in the corners of domain Ω.

Algorithm 2. Training PINN for BVP (3), (4) with quasiclassical loss functional (11)

Data: boundary functions $\chi_1, \varphi_2, \varphi_3, \psi_3, \psi_4$, probability distributions ν_0 and ν_1, initial learning rate α_0, absolute tolerance $\epsilon > 0$;

Result: optimal PINN parameter set $\boldsymbol{\theta}$;

choose random initial parameter set $\boldsymbol{\theta}_0$;

repeat /* `stochastic gradient descent step` */

Generate two random samples for the domain Ω and the boundary $\partial\Omega$, namely:
– generate a random sample $\{(\xi_{i_0}, \eta_{i_0})\}_{i_0=1}^{n_0}$ of n_0 points from Ω with distribution ν_0;
– generate a random sample $\{(\xi_{i_1}, \eta_{i_1})\}_{i_1=1}^{n_1}$ of n_1 points from $\gamma_1 \subset \partial\Omega$ with distribution ν_1;
Calculate the functional $\mathcal{L}_Q(f)$ for the generated random samples combined into a minibatch $\mathbf{s}_k = \left\{\{(\xi_{i_0}, \eta_{i_0})\}_{i_0=1}^{n_0}, \{(\xi_{i_1}, \eta_{i_1})\}_{i_1=1}^{n_1}\right\}$:

$$\mathcal{L}_Q(f, \mathbf{s}_k; \boldsymbol{\theta}_k) \leftarrow \frac{1}{n_0}\sum_{i_0=1}^{n_0}\left(f_\xi^2(\xi_{i_0}, \eta_{i_0}; \boldsymbol{\theta}_k) + f_\eta^2(\xi_{i_0}, \eta_{i_0}; \boldsymbol{\theta}_k) - 2\Psi(\xi_{i_0}) f_\xi(\xi_{i_0}, \eta_{i_0}; \boldsymbol{\theta}_k) - 2\Phi(\eta_{i_0}) f_\eta(\xi_{i_0}, \eta_{i_0}; \boldsymbol{\theta}_k)\right) + \frac{1}{n_1}\sum_{i_1=1}^{n_1}\left(f^2(\xi_{i_1}, \eta_{i_1}; \boldsymbol{\theta}_k) - 2\chi_1(\xi_{i_1}) f(\xi_{i_1}, \eta_{i_1}; \boldsymbol{\theta}_k)\right)$$

Perform a number of gradient descent steps with a minibatch (random points) $\mathbf{s}_k$ using the adaptive Adam algorithm (or other neural network learning algorithm) with learning rate α_k (the learning rate α_k is updated automatically at each step):

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha_k \nabla_{\boldsymbol{\theta}} \mathcal{L}_Q(f, \mathbf{s}_k; \boldsymbol{\theta}_k).$$

until $||\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k|| < \epsilon$; /* `absolute tolerance` */

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{k+1}$;

6. CONCLUSIONS

We derived the loss functional, based on the quasiclassical variational principle, to train a neural network for hyperbolic PDE. The obtained loss functional (11) has obvious advantages over the residuals functional (10) for BVP (3), (4), because the quasiclassical loss functional (11) contains only first-order derivatives of unknown function and requires evaluation of two integrals over the domain Ω and the segment γ_1 of the boundary $\partial\Omega$, while the residual loss functional (10) contains derivatives of unknown function up to the second order and requires evaluation of one integral over the domain Ω and four integrals

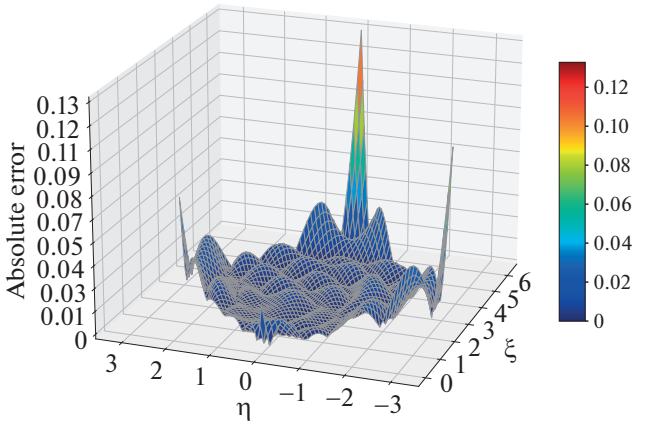

Fig. 11. 3D surface of absolute error of PINN approximation to BVP (3), (7) with quasiclassical loss functional (11).

over the segments $\gamma_1, \gamma_2, \gamma_3, \gamma_4$ of the boundary $\partial\Omega$. The implementation of the neural network training Algorithm 2 with the obtained loss functional (11) demonstrates the convergence of the neural network training process and the achievement of sufficiently high quality indicators of the resulting neural network for the boundary value problem (3), (4).

FUNDING

The research was supported by the RUDN University, project no. 002092-0-000.

CONFLICT OF INTEREST

The author of this work declares that he has no conflicts of interest.

REFERENCES

1. M. Raissi, P. Perdikaris, and G. E. Karniadakis, J. Comput. Phys. **378**, 686 (2019). https://doi.org/10.1016/j.jcp.2018.10.045
2. S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, J. Sci. Comput. **92** (3), 88 (2022). https://doi.org/10.1007/s10915-022-01939-z
3. J. Sirignano and K. Spiliopoulos, J. Comput. Phys. **375**, 1339 (2018). https://doi.org/10.1016/j.jcp.2018.08.029
4. W. E and B. Yu, Communications in Mathematics and Statistics **6**, 1 (2018). https://doi.org/10.1007/s40304-018-0127-z
5. Yi. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, J. Comput. Phys. **394**, 56 (2019). https://doi.org/10.1016/j.jcp.2019.05.024
6. N. Geneva and N. Zabaras, J. Comput. Phys. **403**, 109056 (2020). https://doi.org/10.1016/j.jcp.2019.109056
7. V. M. Filippov, V. M. Savchin, and S. G. Shorokhov, J. Math. Sci. **68**, 275 (1994). https://doi.org/10.1007/bf01252319
8. V. M. Shalov, Dokl. Akad. Nauk SSSR **151**, 292 (1963).
9. V. M. Shalov, Dokl. Akad. Nauk SSSR **151**, 511 (1963).
10. V. M. Shalov, Differ. Uravn. **1**, 1338 (1965).
11. V. M. Filippov, Differ. Uravn. **20**, 1961 (1984).
12. V. M. Filippov and A. N. Skorohodov, Differ. Uravn. **13**, 1113 (1977).
13. V. M. Filippov and A. N. Skorohodov, Differ. Uravn. **13**, 1434 (1977).
14. V. M. Filippov, Dokl. Akad. Nauk SSSR **285**, 302 (1985).
15. V. V. Shelukhin, Soviet Math. Dokl. **43**, 735 (1991).
16. G. P. Paskalev, IOP Conf. Ser.: Mater. Sci. Eng. **618**, 012091 (2019). https://doi.org/10.1088/1757-899x/618/1/012091
17. W. H. Young, Proc. London Math. Soc. **s2–16**, 273 (1917). https://doi.org/10.1112/plms/s2-16.1.273
18. I. M. Sobol', *Numerical Monte Carlo methods* (Nauka, Moscow, 1973).
19. D. P. Kingma and J. Ba, in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, 2015*, Ed. by Y. Bengio and Y. LeCun (2015).
20. M. Abadi, P. Barham, J. Chen, Zh. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, Sh. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Yu. Yu, and X. Zheng, in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (USENIX Association, Savannah, GA, 2016), p. 265. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.