Improving Physics-Informed Neural Networks via Quasiclassical Loss Functionals

Шорохов С.Г.

The 8th International Conference on Deep Learning in Computational Physics

SINP MSU, Moscow, Russia
June 21, 2024

Contents

Филиппов В.М., Савчин В.М., Шорохов С.Г. Вариационные принципы для непотенциальных операторов // Итоги науки и техники. Серия Современные проблемы математики. Новейшие достижения. ВИНИТИ, 1992, Том 40, С.3–176

Filippov V.M., Savchin V.M., Shorokhov S.G. Variational principles for nonpotential operators // Journal of Mathematical Sciences. 1994. Vol.68, No.3, pp. 275-398 https://doi.org/10.1007/BF01252319

Loss functionals for neural network training

Neural network training requires a loss functional: residual functional, energy functional, etc.

Some authors suggest, in the absence of classical (energy) functionals, to use nonclassical functionals from variational principles for nonpotential operators for neural network training:

- Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. Journal of Computational Physics 394 (2019) 56–81
- N. Geneva and N. Zabaras. Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks. Journal of Computational Physics 403 (2020) 109056

The prioblem: can nonclassical variational functionals of the theory of variational principles for nonpotential operators be used in training neural networks that approximate solutions to boundary value problems for equations of mathematical physics?

Physics-Informed Neural Networks (PINNs)

M. Raissi, P. Perdikaris, G.E. Karniadakis. Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations. Journal of Computational Physics 378 (2019) 686–707

Given

- physical model equation (PDE) $\mathscr{L}\left[\mathbf{u}\right] = 0,\ \mathbf{x} \in \Omega$
- initial and boundary conditions $\mathscr{B}\left[\mathbf{u}\right] = 0,\ \mathbf{x} \in \partial\Omega$
- physical laws (properties) (if any) $\mathscr{P}\left[\mathbf{u}\right] = 0,\ \mathbf{x} \in \Omega$

Apprixímate the solution u with a neural network output using the residual loss functional of the form

$$\mathscr{J}\left[\mathbf{u}\right] = \|\mathscr{L}\left[\mathbf{u}\right]\|_{\Omega}^{2} + \|\mathscr{B}\left[\mathbf{u}\right]\|_{\partial\Omega}^{2} + \|\mathscr{P}\left[\mathbf{u}\right]\|_{\Omega}^{2}$$

The method of symmetrizing operator

The method of constructing variational formulations for boundary value problems, proposed by V.M. Shalov in 1963, consists of constructing two vector operators $\mathbf{A}$ and $\mathbf{B}$ such that operator $\mathbf{A}$ is $\mathbf{B}$-symmetric:

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

and $\mathbf{B}$-positive:

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

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

where the boundary value problem (partial differential equation and boundary conditions) is represented in the form

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

where $\mathbf{f}$ is a vector function, then the variational functional for the boundary value problem has the form

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

Boundary value problem for hyperbolic equation

We study the construction of a variational formulation for a homogeneous hyperbolic equation

$$u_{\xi\eta} = 0 \tag{1}$$

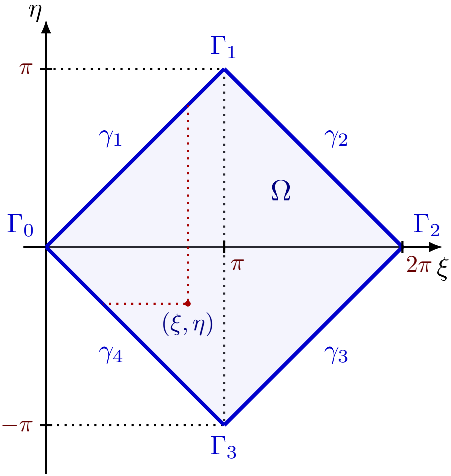
in a rhombus-shaped domain Ω with vertices at the points $\Gamma_0(0, 0)$, $\Gamma_1(\pi, \pi)$, $\Gamma_2(2\pi, 0)$, $\Gamma_3(\pi, -\pi)$ with boundary conditions

$$\begin{cases} u\left|_{\gamma_1}\right. = \chi_1(\xi), \\ u_\eta\left|_{\gamma_2}\right. = \varphi_2(\eta), \\ u_\eta\left|_{\gamma_3}\right. = \varphi_3(\eta), \\ u_\xi\left|_{\gamma_3}\right. = \psi_3(\xi), \\ u_\xi\left|_{\gamma_4}\right. = \psi_4(\xi), \end{cases} \tag{2}$$

where the segment γ_1 connects the points $\Gamma_0(0, 0)$ and $\Gamma_1(\pi, \pi)$, the segment γ_2 connects the points $\Gamma_1(\pi, \pi)$ and $\Gamma_2(2\pi, 0)$, the segment γ_3 connects the points $\Gamma_2(2\pi, 0)$ and $\Gamma_3(\pi, -\pi)$, the segment γ_4 connects the points $\Gamma_3(\pi, -\pi)$ and $\Gamma_0(0, 0)$.

Here $u \in W_2^1(\Omega)$ and $(\chi_1, \varphi_2, \varphi_3, \psi_3, \psi_4) \in L_2(\gamma_1 \times \gamma_2 \times \gamma_3 \times \gamma_3 \times \gamma_4)$.

Domain Ω and outer normal to $\partial\Omega$



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 calculated using the formulas:

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

$$\vec{n}\,(\gamma_2) = \frac{1}{\sqrt{2}}\,(1,\, 1)\,,$$

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

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

Vector operators $\mathbf{A}$ and $\mathbf{B}$ and vector function $\mathbf{f}$

The vector differential operator $\mathbf{A}$ and the vector integro-differential operator $\mathbf{B}$, acting from $W_2^1(\Omega)$ to $L_2(\Omega \times \gamma_1 \times \gamma_2 \times \gamma_3 \times \gamma_3 \times \gamma_4)$, are defined by the equalities

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

and the vector function $\mathbf{f}$ is equal to

$$\mathbf{f} = \begin{bmatrix} 0 \\ \chi_1 \\ \varphi_2 \\ \varphi_3 \\ \psi_3 \\ \psi_4 \end{bmatrix}$$

Scalar product on $\Omega\times\gamma_1\times\gamma_2\times\gamma_3\times\gamma_3\times\gamma_4$

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

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

$$+ \int\limits_{\gamma_3} (\mathbf{A}\,u)_4\,(\mathbf{B}\,v)_4\; ds + \int\limits_{\gamma_3} (\mathbf{A}\,u)_5\,(\mathbf{B}\,v)_5\; ds + \int\limits_{\gamma_4} (\mathbf{A}\,u)_6\,(\mathbf{B}\,v)_6\; ds$$

The variational functional by V.M. Shalov for the boundary value problem under consideration (1)–(2) has the form

$$D\left[u\right] = \int\limits_{\Omega} \left(u_\xi^2 + u_\eta^2\right)\, d\xi d\eta + \int\limits_{\gamma_1} u^2\, ds +$$

$$-2\int\limits_{\gamma_1} \chi_1 u\, ds + 2\int\limits_{\gamma_2} \varphi_2 n_1 \int\limits_{\xi}^{\gamma_1} u_\eta\left(\zeta,\eta\right) d\zeta\, ds + 2\int\limits_{\gamma_3} \varphi_3 n_1 \sin\xi \int\limits_{\xi}^{\gamma_4} u_\eta\left(\zeta,\eta\right) d\zeta\, ds +$$

$$+2\int\limits_{\gamma_3} \psi_3 n_2 \int\limits_{\eta}^{\gamma_2} u_\xi\left(\xi,\tau\right) d\tau\, ds + 2\int\limits_{\gamma_4} \psi_4 n_2 \int\limits_{\eta}^{\gamma_1} u_\xi\left(\xi,\tau\right) d\tau\, ds. \qquad (3)$$

However, this form of the functional is not convinient for use as a loss functional in training a neural network due to repeated integrals.

Final form of variational functional

Theorem

The variational functional by V.M. Shalov (3) for the boundary value problem (1)–(2) can be written in the form

$$D\left[u\right] = \int\limits_{\Omega} \left(u_{\xi}^{2} + u_{\eta}^{2} - 2\Phi\, u_{\eta} - 2\Psi\, u_{\xi}\right)\, d\xi d\eta + \int\limits_{\gamma_1} \left(u^{2} - 2\chi_1 u\right) ds, \tag{4}$$

where

$$\Phi\left(\eta\right) = \begin{cases} \varphi_2\left(\eta\right), & 0 \leqslant \eta \leqslant \pi, \\ \varphi_3\left(\eta\right), & -\pi \leqslant \eta < 0, \end{cases}, \quad \Psi\left(\xi\right) = \begin{cases} \psi_3\left(\xi\right), & \pi \leqslant \xi \leqslant 2\pi, \\ \psi_4\left(\xi\right), & 0 \leqslant \xi < \pi. \end{cases}$$

Functional (4) contains the first-order derivatives of the function u and can be used for training a neural network that approximates the solution of the boundary value problem (1)-(2).

Neural network model of boundary value problem

The solution of a boundary value problem for hyperbolic equation

$$u_{\xi\eta} = 0,\ (\xi,\ \eta) \in \Omega$$

with boundary conditions

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

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

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

Training with residual functional

When training a neural network, the residual functional can be used as a loss (error) functional:

$$\mathcal{L}_R(f) = \|f_{\xi\eta}\|_\Omega^2 + \|f - \chi_1\|_{\gamma_1}^2 + \|f_\eta - \varphi_2\|_{\gamma_2}^2 +$$

$$+ \|f_\eta - \varphi_3\|_{\gamma_3}^2 + \|f_\xi - \psi_3\|_{\gamma_3}^2 + \|f_\xi - \psi_4\|_{\gamma_4}^2 ,$$

where $\|f\|_X^2 = \int_X |f(x)|^2 \rho(x)\, dx$, $\rho(x)$ is the density of some probability distribution on X (for example, a uniform distribution).

When using the residual functional $\mathcal{L}_R(f)$, it is necessary to calculate five integrals included in $\mathcal{L}_R(f)$, for which it is necessary to construct random samples from five different domains – Ω, γ_1, γ_2, γ_3, γ_4, and calculate the partial derivatives of the unknown function f up to and including the second order.

Training with quasiclassical functional

When training a neural network, the obtained quasiclassical functional can also be used as a loss (error) functional.

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

When using the quasiclassical functional $\mathcal{L}_Q(f)$, it is necessary to calculate two integrals from $\mathcal{L}_Q(f)$, and, accordingly, to construct two random samples from the domain Ω and from the part of the boundary γ_1, and calculate the first-order partial derivatives of the unknown function f.

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

The algorithm for training a neural network using a quasiclassical functional

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

as a loss functional includes the following steps:

1) Select the initial set of neural network parameters $\boldsymbol{\theta}_0$ and the initial learning rate α_0
2) 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 the domain 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 the boundary segment $\gamma_1 \subset \partial\Omega$ with distribution ν_1

Training with quasiclassical functional $\mathcal{L}_Q$ (2)

3) Calculate the functional $\mathcal{L}_Q(f)$ for the generated random samples combined into a mini-batch $\mathbf{s}_k = \left\{ \{(\xi_{i_0}, \eta_{i_0})\}_{i_0=1}^{n_0}, \{(\xi_{i_1}, \eta_{i_1})\}_{i_1=1}^{n_1} \right\}$:

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

$$\left. -2\,\Psi(\xi_{i_0})\, f_\xi(\xi_{i_0}, \eta_{i_0}; \boldsymbol{\theta}_k) - 2\,\Phi(\eta_{i_0})\, f_\eta(\xi_{i_0}, \eta_{i_0}; \boldsymbol{\theta}_k) \right) +$$

$$+\frac{1}{n_1} \sum_{i_1=1}^{n_1} \left(f^2(\xi_{i_1}, \eta_{i_1}; \boldsymbol{\theta}_k) - 2\,\chi_1(\xi_{i_1})\, f(\xi_{i_1}, \eta_{i_1}; \boldsymbol{\theta}_k) \right)$$

4) Perform a number of gradient descent steps with a mini-batch (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):

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

5) Repeat steps 2-4 until the change in neural network parameters $\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\|$ becomes small enough (or use another stopping criterion)

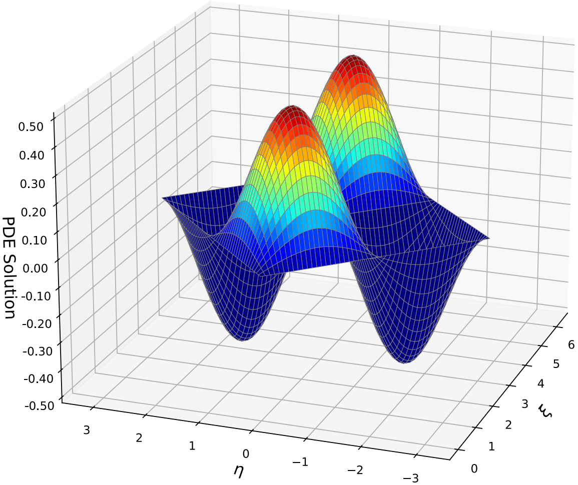
Neural network for a boundary value problem

We run computational experiments on training neural networks to approximate the solution of the hyperbolic equation $u_{\xi\eta} = 0$ with boundary conditions

$$\begin{cases} u\,|_{\gamma_1} = 0, \\ u_\eta\,|_{\gamma_2} = -\dfrac{1}{2}\sin 2\eta, \\ u_\eta\,|_{\gamma_3} = -\dfrac{1}{2}\sin 2\eta, \\ u_\xi\,|_{\gamma_3} = \dfrac{1}{2}\sin 2\xi, \\ u_\xi\,|_{\gamma_4} = \dfrac{1}{2}\sin 2\xi, \end{cases}$$

which is equal to

$$u(\xi, \eta) = -\frac{1}{4}\cos 2\xi + \frac{1}{4}\cos 2\eta$$



Parameters of computational experiments

We use the residual functional and the constructed quasiclassical functional as a loss functional.

Computational experiments are carried out for the neural network architecture specified below with the following hyperparameter values:

- a feedforward neural network (FF) with dense layers is used
- the number of hidden layers is 4 with the activation function tanh
- the number of neurons in the hidden layers is 100
- one neuron without the activation function is in the output layer
- 200 training epochs are performed with 10 SGD steps for each epoch
- the Adam optimizer is used with an initial learning rate of 0.0001
- a batch contains 1000 samples from domain Ω and 500 samples for each section of the boundary $\partial\Omega$

The program code is implemented using TensorFlow framework and is executed on a MacBook Pro computer with M2Max processor.

Residual functional – the sampler (1)

```python
# %% Sampling function for residuals functional - randomly sample xi-eta pairs
def sampler_r(nSim_i, nSim_b):
    ''' Sample xi-eta points from the function's domain;
        points are sampled uniformly on the interior of the domain
        and for initial/terminal/boundary points

    Args:
        nSim_i: number of points in the interior of domain to sample
        nSim_b: number of points at the boundary to sample
    '''
    # Sample #0: domain interior
    x_0 = np.random.uniform(low=x_low, high=x_high, size=[nSim_i, 1])
    t_0 = np.random.uniform(low=t_low, high=t_high, size=[nSim_i, 1])

    # Sample #1: initial condition (gamma1 curve)
    x_1 = np.random.uniform(low=x_low, high=x_high, size=[nSim_b, 1])
    t_1 = t_low * np.ones((nSim_b, 1)) # np.zeros((nSim_b, 1))
```

...

Residual functional – the sampler (2)

...

```
# Sample #2: boundary condition (gamma2 curve)
x_2 = x_high * np.ones((nSim_b, 1))
t_2 = np.random.uniform(low=t_low, high=t_high, size=[nSim_b, 1])

# Sample #3: terminal condition (gamma3 curve)
x_3 = np.random.uniform(low=x_low, high=x_high, size=[nSim_b, 1])
t_3 = t_high * np.ones((nSim_b, 1))

# Sample #4: boundary condition (gamma4 curve)
x_4 = x_low * np.ones((nSim_b, 1)) # np.zeros((nSim_b, 1))
t_4 = np.random.uniform(low=t_low, high=t_high, size=[nSim_b, 1])

return x_0 + t_0, x_0 - t_0, x_1 + t_1, x_1 - t_1, \
    x_2 + t_2, x_2 - t_2, x_3 + t_3, x_3 - t_3, \
    x_4 + t_4, x_4 - t_4
```

Residual functional – loss calculation (1)

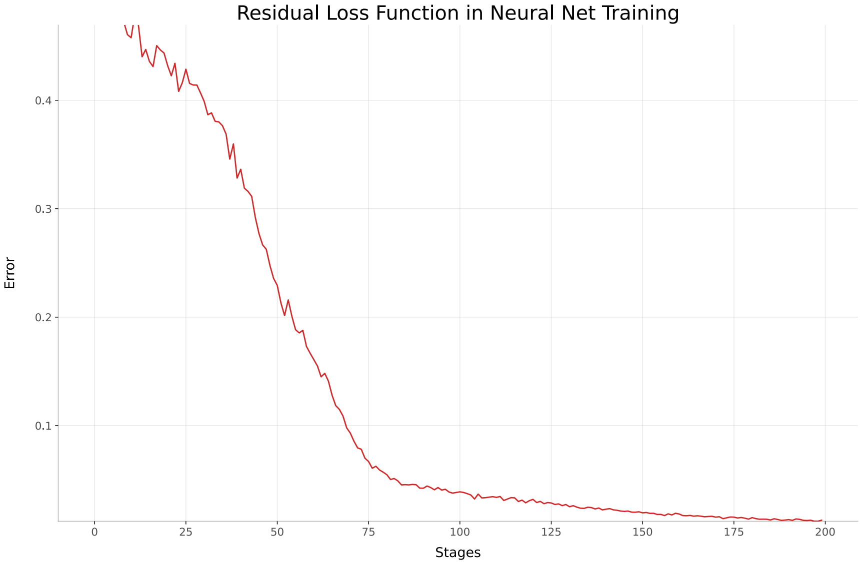
```
# Residuals loss functional for hyperbolic PDE boundary problem
def loss_r(model, x0, e0, x1, e1, x2, e2, x3, e3, x4, e4):
    ''' Compute total loss for training the neural network

    Args:
        model:      neural network model object
        x0:         sampled xi points in the interior
        e0:         sampled eta points in the interior
        x1,...,x4: sampled xi points at the boundary
        e1,...,e4: sampled eta points at the boundary

    '''
    # Loss term #0: average L2-norm of hyperbolic PDE differential operator
    # function value and derivatives at sampled points
    with tf.GradientTape() as tU0x:
        with tf.GradientTape() as tU0:
            U0 = model(x0, e0)
        U0x = tU0.gradient(U0, x0)
    U0xe = tU0x.gradient(U0x, e0)

    L0 = tf.reduce_mean( tf.square(U0xe) )
```

...

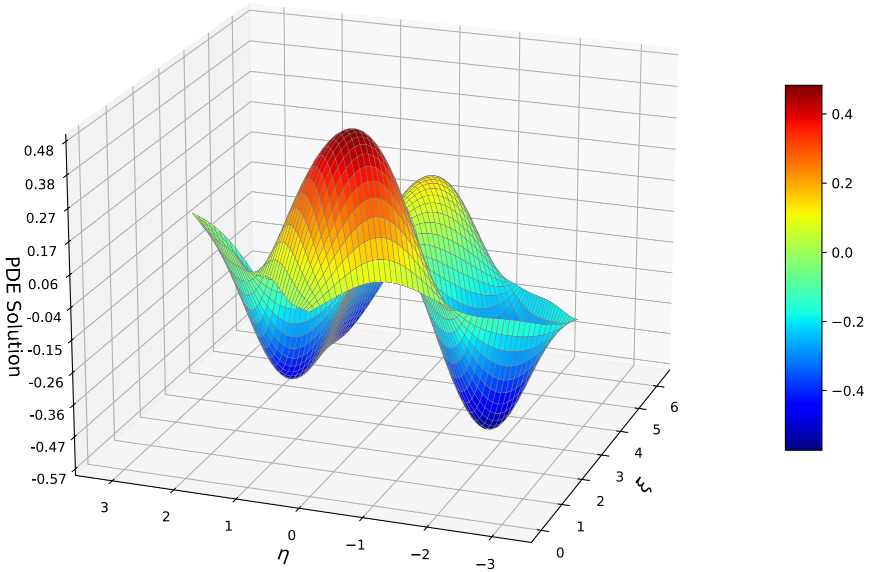
Residual functional – loss calculation (2)

```
# Loss term #1: average L2-norm of initial condition (on gamma1)
U1 = model(x1, e1)
L1 = tf.reduce_mean( tf.square(U1) )

# Loss term #2: average L2-norm of boundary condition (on gamma2)
with tf.GradientTape() as tU2:
    U2 = model(x2, e2)
U2e  = tU2.gradient(U2, e2)
L2 = tf.reduce_mean( tf.square( U2e + 0.5*tf.math.sin(2.*e2) ) )

# Loss term #3: average L2-norm of terminal condition (on gamma3)
with tf.GradientTape() as tU3:
    U3 = model(x3, e3)
U3x, U3e  = tU3.gradient(U3, [x3, e3])
L3 = tf.reduce_mean( tf.square( U3e + 0.5*tf.math.sin(2.*e3) ) ) + \
     tf.reduce_mean( tf.square( U3x - 0.5*tf.math.sin(2.*x3) ) )

# Loss term #4: average L2-norm of boundary condition (on gamma4)
with tf.GradientTape() as tU4:
    U4 = model(x4, e4)
U4x  = tU4.gradient(U4, x4)
L4 = tf.reduce_mean( tf.square( U4x - 0.5*tf.math.sin(2.*x4) ) )

return L0 + L1 + L2 + L3 + L4 #, L0, L1, L2, L3, L4
```

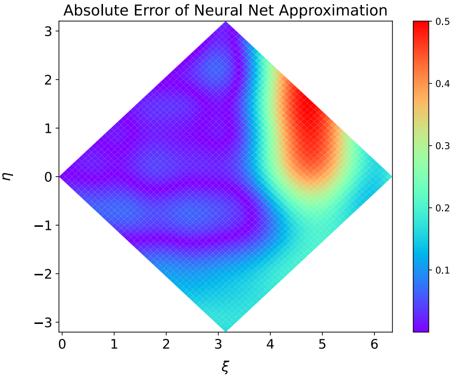
Residual functional – the loss graph

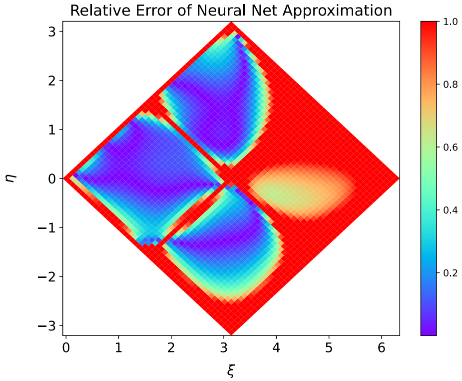


Residual functional – the approximation

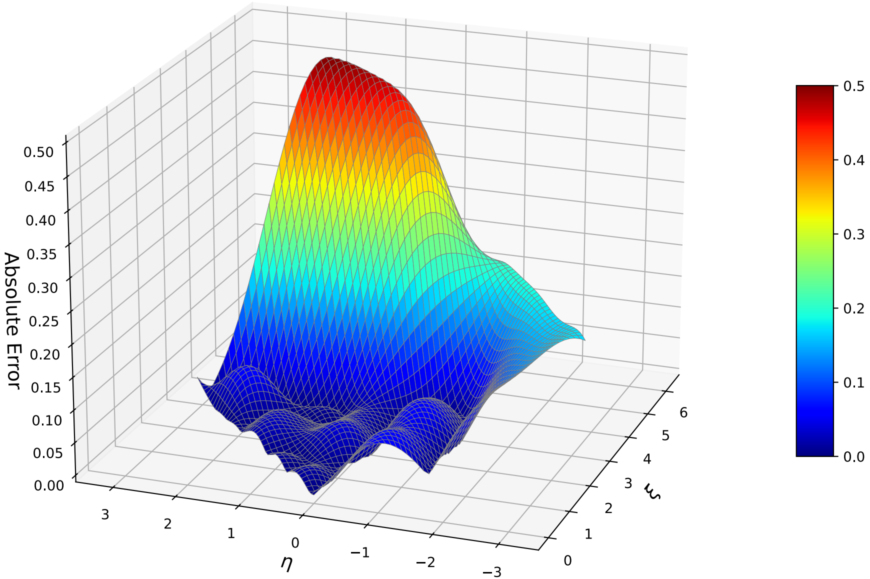


Residual functional – 2d error





Residual functional – 3d error surface



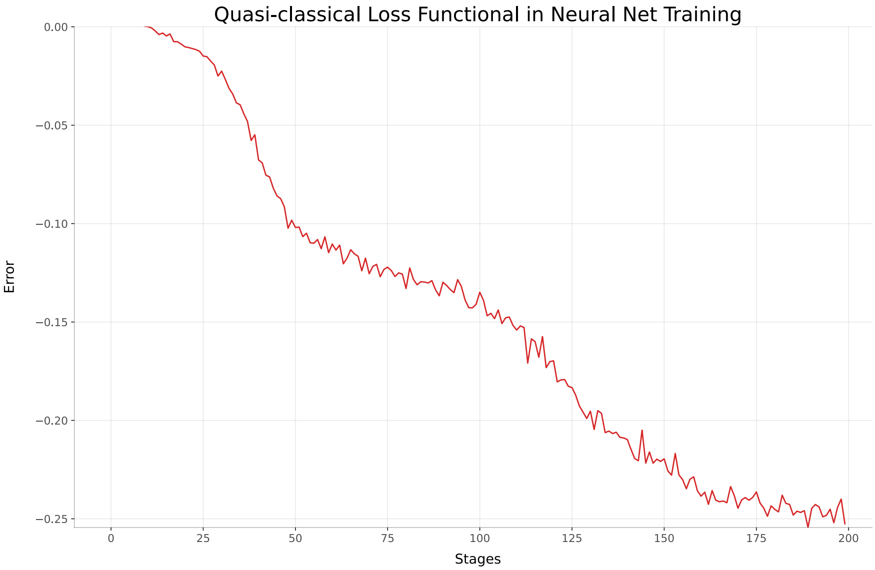
Quasiclassical functional – the sampler

```python
# %% Sampling function for quasi-classical functional - randomly sample xi-eta pairs
def sampler_q(nSim_i, nSim_b):
    ''' Sample xi-eta points from the function's domain;
        points are sampled uniformly on the interior of the domain
        and for initial/terminal/boundary points

    Args:
        nSim_i: number of points in the interior of domain to sample
        nSim_b: number of points at the boundary to sample
    '''
    # Sample #0: domain interior
    x_0 = np.random.uniform(low=x_low, high=x_high, size=[nSim_i, 1])
    t_0 = np.random.uniform(low=t_low, high=t_high, size=[nSim_i, 1])

    # Sample #1: initial condition (gamma1 curve)
    x_1 = np.random.uniform(low=x_low, high=x_high, size=[nSim_b, 1])
    t_1 = t_low * np.ones((nSim_b, 1)) # np.zeros((nSim_b, 1))

    return x_0 + t_0, x_0 - t_0, x_1 + t_1, x_1 - t_1
```

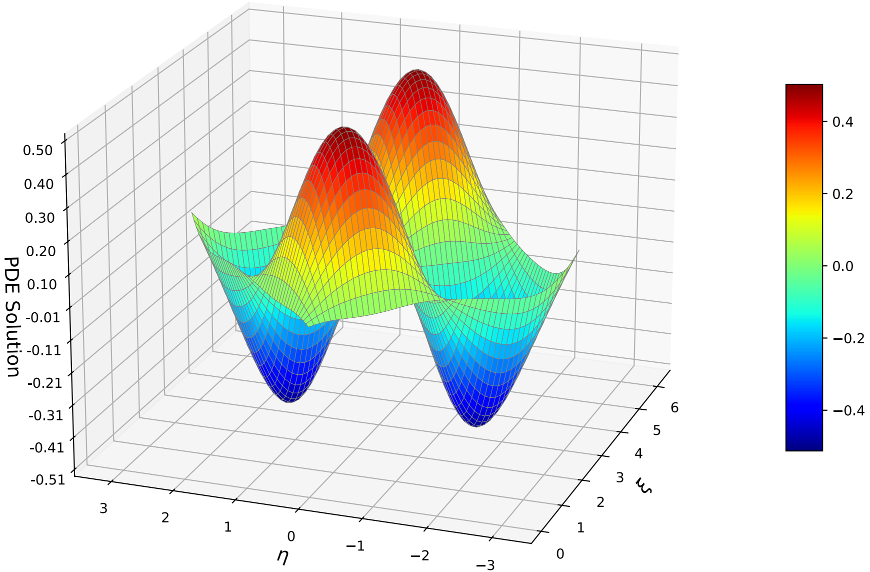
Quasiclassical functional – loss calculation

```python
# quasi-classical loss functional for hyperbolic PDE boundary problem
def loss_q(model, x0, e0, x1, e1):
    ''' Compute total loss for training the neural network
    Args:
        model:       neural network model object
        x0:         sampled xi points in the interior
        e0:         sampled eta points in the interior
        x1:         sampled xi points at the boundary
        e1:         sampled eta points at the boundary
    '''
    # Loss term #0: average L2-norm of hyperbolic PDE differential operator
    # function value and derivatives at sampled points
    with tf.GradientTape() as tU0:
        U0 = model(x0, e0)
    U0x, U0e  = tU0.gradient(U0, [x0, e0])
    L0 = tf.reduce_mean(tf.square(U0x) + tf.square(U0e) - \
                        U0x*tf.math.sin(2.*x0) + U0e*tf.math.sin(2.*e0))

    # Loss term #1: average L2-norm of initial condition (on gamma1)
    U1 = model(x1, e1)
    L1 = tf.reduce_mean( tf.square(U1) )

    return L0 + L1 #, L0, L1
```

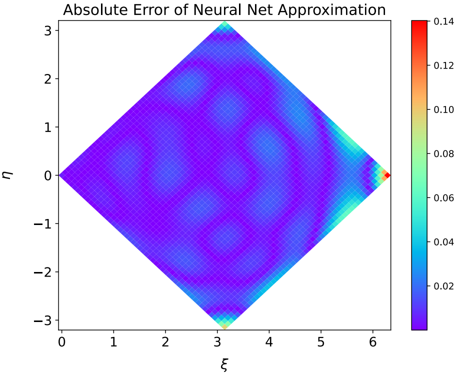
Quasiclassical functional – the loss graph

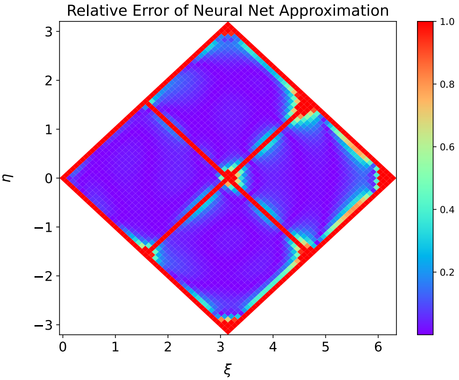


Quasiclassical functional – the approximation

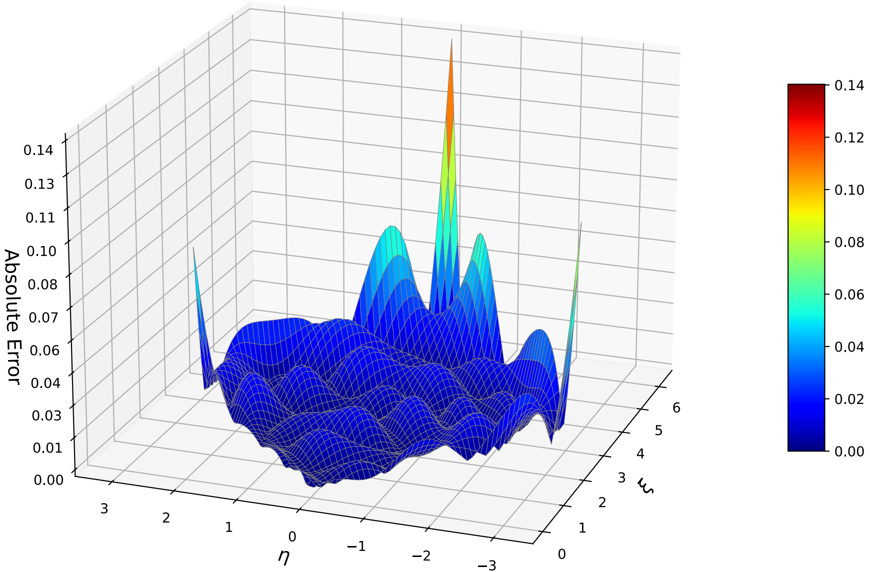


Quasiclassical functional – 2d error





Quasiclassical functional – 3d error surface



Indicators of training and quality	Neural network with residual functional	Neural network with quasiclassical functional
Training time (for 200 epochs)	119.9 sec	40.2 sec
MSE (mean squared error)	0.0305	0.0002
MAE (mean absolute error)	0.1169	0.0090
R^2(coefficient of determination)	49.2%	99.7%

Conclusion

- The boundary value problem (1)–(2) under consideration for hyperbolic equation admits variational functional

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

  which can be used as a loss functional when training a neural network
- The obtained variational functional has a number of advantages over the residual functional when training a physics-informed neural network.
- The implementation of the neural network training algorithm with the obtained variational functional demonstrates the convergence of the neural network training process and the achievement of sufficiently high quality indicators of the resulting physics-informed neural network for the boundary value problem (1)–(2)

Any question?