Report: Deep Ritz Method for Solving the Biharmonic Equation with Neumann-Type Boundary Conditions (Phase-2)

1. Introduction

The biharmonic equation is a fourth-order partial differential equation (PDE) that arises in various applications in physics and engineering.

The general biharmonic problem is stated as:

$$\boldsymbol{\Delta^2 u = f \text{ in } \Omega}$$

where $\Delta^2 = \Delta(\Delta u)$is the biharmonic operator, obtained by applying the Laplacian operator twice. For a two-dimensional domain $\Omega \subset \mathbb{R}^2$, this expands to:

$$\Delta^2 u = \frac{\partial^4 u}{\partial x_1^4} + 2\frac{\partial^4 u}{\partial x_1^2 \partial x_2^2} + \frac{\partial^4 u}{\partial x_2^4}$$

The Deep Ritz Method:

The Deep Ritz Method (DRM) is a physics-informed machine learning technique that combines the classical Ritz variational method with deep neural networks. Instead of discretizing the domain and using finite-dimensional function spaces, DRM parameterizes the solution using a neural network and minimizes the energy functional directly.

Key advantages of DRM:

1. Automatic differentiation: Neural network frameworks provide derivatives automatically
2. Flexibility: Can handle complex geometries and high-dimensional problems
3. Universal approximation: Neural networks can approximate arbitrary smooth functions

2. Problem Formulation

2.1 The Biharmonic Problem (P2)

We consider the biharmonic equation in the unit square $\Omega = (0,1)^2$:

$$\boldsymbol{\Delta^2 u = f \text{ in } \Omega}$$

with mixed boundary conditions:

- Essential (Dirichlet): $u = g_1$on $\partial\Omega$
- Neumann-type: $\frac{\partial^2 u}{\partial n^2} = g_2$on $\partial\Omega$

The second normal derivative is defined as:

$$\frac{\partial^2 u}{\partial n^2} = \mathrm{n}^T \cdot D^2 u \cdot \mathrm{n} = n_1^2 \frac{\partial^2 u}{\partial x_1^2} + 2n_1 n_2 \frac{\partial^2 u}{\partial x_1 \partial x_2} + n_2^2 \frac{\partial^2 u}{\partial x_2^2}$$

where $\mathrm{n} = (n_1, n_2)$is the outward normal vector and $D^2 u$is the Hessian matrix of second derivatives.

2.2 Ritz Variational Formulation

The classical Ritz method seeks the solution by minimizing an energy functional. For Problem (P2), the energy functional is:

$$\mathcal{L}(v) = \frac{1}{2}\int_\Omega \| D^2 v \|_F^2 \, dx - \int_\Omega f v \, dx - \int_{\partial\Omega} g_2 \frac{\partial v}{\partial n} \, ds$$

where:

- $\| D^2 v \|_F^2$is the Frobenius norm squared of the Hessian matrix
- The last term incorporates the natural (Neumann-type) boundary condition

The Frobenius norm squared of the Hessian is:

$$\| D^2 v \|_F^2 = (\frac{\partial^2 v}{\partial x_1^2})^2 + 2(\frac{\partial^2 v}{\partial x_1 \partial x_2})^2 + (\frac{\partial^2 v}{\partial x_2^2})^2$$

Admissible space: Classically, we minimize over $K = \{v \in H^2(\Omega): v = g_1 \text{ on } \partial\Omega\}$, where $H^2(\Omega)$is the Sobolev space of functions with square-integrable second derivatives.

Euler-Lagrange equation: The minimizer of $\mathcal{L}(v)$satisfies the original PDE and boundary conditions.

2.3 Penalized Deep Ritz Formulation

In the Deep Ritz Method, enforcing the essential boundary condition $u = g_1$exactly is challenging when using neural networks. Instead, we use a penalty method by adding a penalty term to the energy functional:

$$\mathcal{L}_\lambda(v) = \frac{1}{2}\int_\Omega \| D^2 v \|_F^2 \, dx - \int_\Omega f v \, dx - \int_{\partial\Omega} g_2 \frac{\partial v}{\partial n} \, ds + \frac{\lambda}{2}\int_{\partial\Omega} (v - g_1)^2 \, ds$$

where:

- $\lambda > 0$is the penalty parameter
- The penalty term $\frac{\lambda}{2}\int_{\partial\Omega}\ (v - g_1)^2\ ds$ enforces $v \approx g_1$on the boundary
- Larger λenforces the boundary condition more strongly

Key insight: As $\lambda \to \infty$, the penalized solution converges to the constrained solution.

Admissible space: Now we minimize over $K_\lambda = H^2(\Omega)$(no explicit boundary constraint).

2.4 Monte Carlo Approximation

The integrals in $\mathcal{L}_\lambda$cannot be computed analytically for neural networks. We approximate them using Monte Carlo sampling:

$$\int_\Omega h(x)\ dx \approx \frac{|\ \Omega\ |}{N}\sum_{i=1}^{N}\ h(X_i), X_i \sim iid\mathcal{U}(\Omega)$$

$$\int_{\partial\Omega} h(x)\ ds \approx \frac{|\ \partial\Omega\ |}{M}\sum_{j=1}^{M}\ h(Y_j), Y_j \sim iid\mathcal{U}(\partial\Omega)$$

For the unit square: $|\ \Omega\ | = 1$, $|\ \partial\Omega\ | = 4$.

Empirical loss function:

$$\hat{\mathcal{L}}_\lambda(u_\theta) = \frac{1}{N}\sum_{i=1}^{N}\ \frac{1}{2}\|\ D^2u_\theta(X_i)\ \|_F^2 - f(X_i)u_\theta(X_i) + \frac{4}{M}\sum_{j=1}^{M}[-g_2(Y_j)\frac{\partial u_\theta}{\partial n}(Y_j) + \frac{\lambda}{2}(u_\theta(Y_j) - g_1(Y_j))^2]$$

3. Neural Network Setup

3.1 Architecture

The solution $\mathrm{u}(\mathrm{x}_1, \mathrm{x}_2)$is approximated by a fully connected feed-forward neural network (multi-layer perceptron), implemented in the class U_FCN. The architecture used in this work consists of:

- Input layer: 2 neurons (coordinates $\mathrm{x}_1, \mathrm{x}_2$)
- Hidden layers: 6 layers with widths

$$[\mathbf{64}, \mathbf{128}, \mathbf{256}, \mathbf{512}, \mathbf{256}, \mathbf{128}, \mathbf{64}]$$

- Activation function: hyperbolic tangent ($\tanh$) after each hidden layer

- Output layer: 1 neuron representing $u_\theta(x_1, x_2)$

Let $x = (x_1, x_2) \in \mathbb{R}^2$. If we denote the hidden layer widths by $(h_1, h_2, \dots, h_6) = (64,128,256,512,256,128,64)$ and the activation function by $\sigma(z) = \tanh(z)$, the network can be written as:

$$\begin{aligned} z_1 &= \sigma(W_1 x + b_1), \\ z_2 &= \sigma(W_2 z_1 + b_2), \\ z_3 &= \sigma(W_3 z_2 + b_3), \\ z_4 &= \sigma(W_4 z_3 + b_4), \\ z_5 &= \sigma(W_5 z_4 + b_5), \\ z_6 &= \sigma(W_6 z_5 + b_6), \\ u_\theta(x) &= W_7 z_6 + b_7, \end{aligned}$$

where

$$\theta = \{W_1, b_1, \dots, W_7, b_7\}$$

is the set of all trainable parameters.

Total number of parameters: **345665**

Choice of activation:

The activation function $\tanh$ is chosen because:

1. It is smooth and infinitely differentiable.
2. It allows stable computation of second-order and fourth-order derivatives using automatic differentiation.
3. ReLU is not suitable for high-order PDEs, as its second derivative vanishes almost everywhere.

3.2 Sampling Strategy

Training requires interior points to enforce the PDE and boundary points to enforce boundary conditions.

Interior sampling:

We draw N points uniformly inside the computational domain

$$\Omega = (0,1)^2,$$

as

$$X_i \sim \mathcal{U}([0,1]^2), i = 1, \dots, N.$$

Typical choice: $N = 1024$ interior points per training batch.

Boundary sampling:

The boundary $\partial\Omega$consists of four edges. We draw $M/4$points uniformly on each edge (with total boundary points $M = 256$). For each edge, we associate the outward unit normal vector:

- Bottom edge ($x_2 = 0$): $n = (0, -1)$
- Right edge ($x_1 = 1$): $n = (1,0)$
- Top edge ($x_2 = 1$): $n = (0,1)$
- Left edge ($x_1 = 0$): $n = (-1,0)$

Resampling:

- During Adam optimization, interior and boundary points are resampled every iteration.
  This reduces overfitting and improves generalization.
- During L-BFGS optimization, a fixed set of points is used for stability.

3.3 Automatic Differentiation

PyTorch's automatic differentiation (torch.autograd.grad) is used to compute all derivatives of the neural network output $u_\theta(x)$with respect to the spatial coordinates.

First derivatives (gradient)

$$\nabla u_\theta(x) = \left(\frac{\partial u_\theta}{\partial x_1}(x), \frac{\partial u_\theta}{\partial x_2}(x)\right).$$

Second derivatives (Hessian)

The Hessian matrix is:

$$H(u_\theta)(x) = \begin{bmatrix} u_{x_1x_1} & u_{x_1x_2} \\ u_{x_2x_1} & u_{x_2x_2} \end{bmatrix}.$$

$u_{x_1}$and u_{x_2}again with respect to x.

Biharmonic operator

The biharmonic operator required in problem (P2) is

$$\Delta^2 u_\theta = \frac{\partial^2}{\partial x_1^2}(\Delta u_\theta) + \frac{\partial^2}{\partial x_2^2}(\Delta u_\theta),$$

and is computed using repeated applications of torch.autograd.grad in the function biharmonic_operator.

Normal derivatives:

For Neumann-type boundary conditions, we require:

- First normal derivative:

$$\frac{\partial u_\theta}{\partial n} = \nabla u_\theta \cdot n$$

- Second normal derivative:

$$\frac{\partial^2 u_\theta}{\partial n^2} = \nabla\left(\frac{\partial u_\theta}{\partial n}\right) \cdot n$$

All derivative computations use create_graph=True so that higher-order derivatives remain part of the computational graph.

4. Loss Function and Deep Ritz Formulation for Problem (P2)

For the biharmonic equation

$$\Delta^2 u = f \text{in } \Omega = (0,1)^2,$$

with boundary conditions

$$u = g_1 \text{on } \partial\Omega, \frac{\partial^2 u}{\partial n^2} = g_2 \text{on } \partial\Omega,$$

we adopt a penalized Deep Ritz formulation.

The total loss consists of:

$$\mathcal{L}_\theta = \mathcal{L}_{\text{interior}} + \mathcal{L}_{\text{boundary}},$$

and in our implementation, we additionally apply uncertainty-based weighting through learnable log-variances.

4.1 Interior Loss

In the Deep Ritz Method, the biharmonic equation is obtained as the Euler–Lagrange equation of the energy functional

$$E(u) = \int_{\Omega} (\frac{1}{2} \mid \nabla^2 u \mid^2 - fu)\ dx.$$

The discrete interior loss implemented in the code is:

$$\mathcal{L}_{\text{interior}} = \frac{1}{\mid \Omega \mid} \sum_{i=1}^{N} (\frac{1}{2} \parallel H(u_\theta(x_i)) \parallel_F^2 - f(x_i) u_\theta(x_i)),$$

where:

- $H(u_\theta)$is the Hessian matrix,
- $\|\cdot\|_F$is the Frobenius norm,
- $f(x)$is the source term,
- $u_\theta(x)$is the neural network prediction.

4.2 Boundary Loss

For problem (P2), the boundary conditions are:

1. **Dirichlet:**

$$u_\theta(x) = g_1(x),$$

2. **Second normal derivative (Neumann-type):**

$$\frac{\partial^2 u_\theta}{\partial n^2} = g_2(x).$$

The penalty-based boundary loss is:

$$\mathcal{L}_{\text{boundary}} = \lambda_D \parallel u_\theta - g_1 \parallel_{\partial\Omega}^2 - \int_{\partial\Omega} g_2(x)\ \frac{\partial u_\theta}{\partial n}(x)\ ds.$$

4.3 Uncertainty-Based Loss Weighting

We introduce learnable log-variances:

- log_var_interior & log_var_boundary

and define the total loss as:

$$\mathcal{L}_\theta = \frac{1}{2}[e^{-s_1}\mathcal{L}_{\text{interior}} + s_1 + e^{-s_2}\mathcal{L}_{\text{boundary}} + s_2],$$

where $s_1 = \text{log\_var\_interior}$ and $s_2 = \text{log\_var\_boundary}$.

5. Training Procedure

5.1 Adam Training (Stochastic)

For epochs $1 \rightarrow E_{\text{switch}}$:

- At every epoch, new interior and boundary samples are drawn.
- Adam optimizer updates the parameters.
- Uncertainty parameters (log_var_*) are updated together with the network.
- Losses and validation errors are logged.

6. Error Computation and Validation

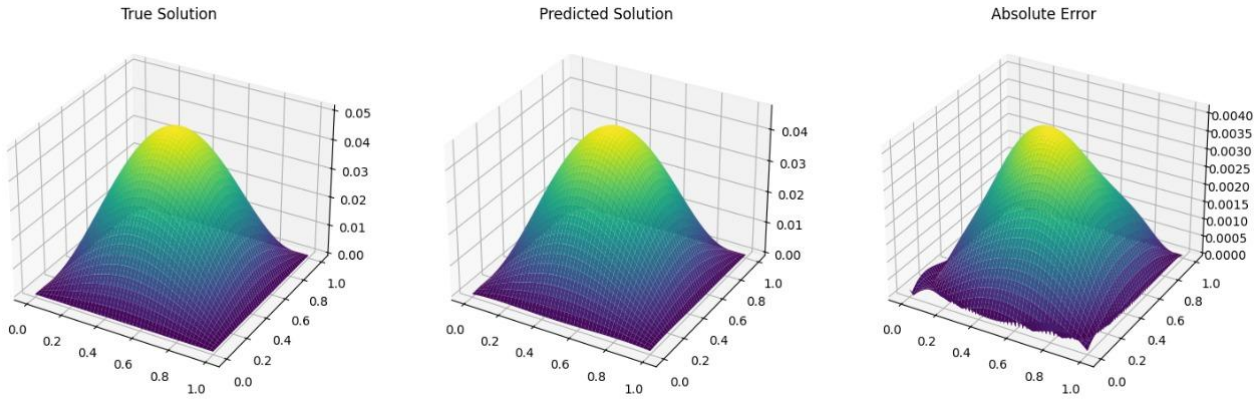
To evaluate accuracy, we compute the **L² error** over a dense set of points:

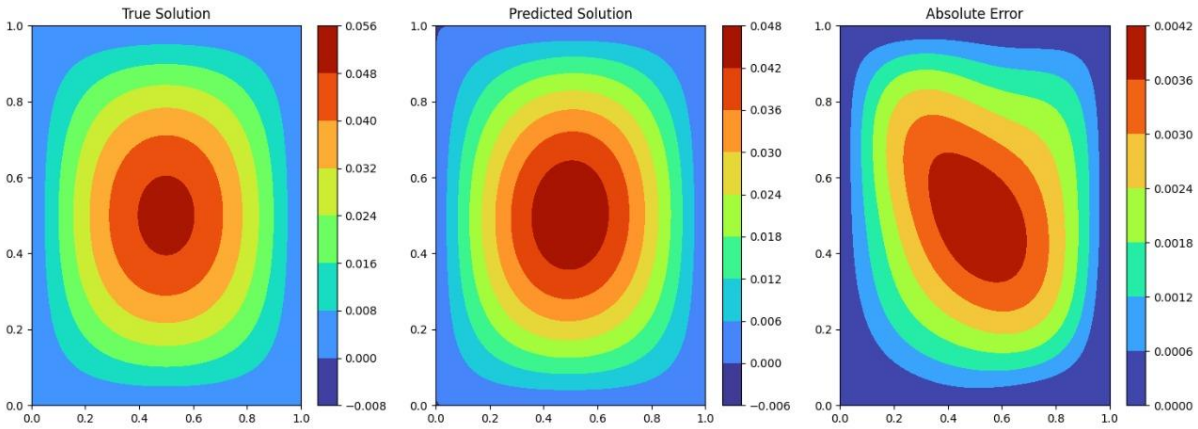
$$\| u_\theta - u \|_{L^2} \approx \sqrt{\frac{1}{N}\sum_{i=1}^{N} (u_\theta(x_i) - u(x_i))^2}$$

7. Visualization of Results

We visualize:

Results Example 3.1





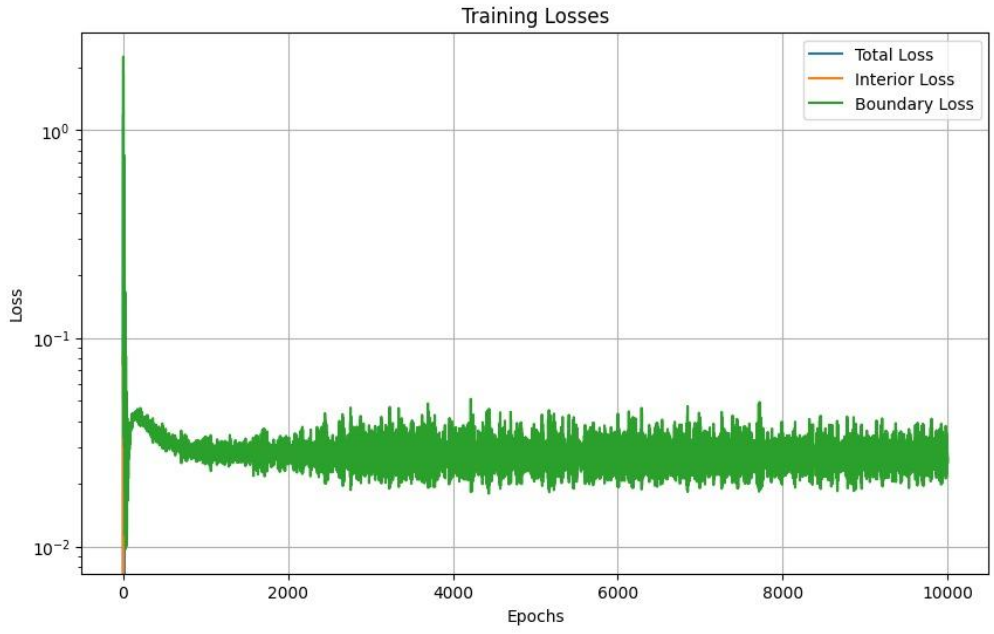
L2: 0.002134957816451788

L2 relative: 0.08436878485050284

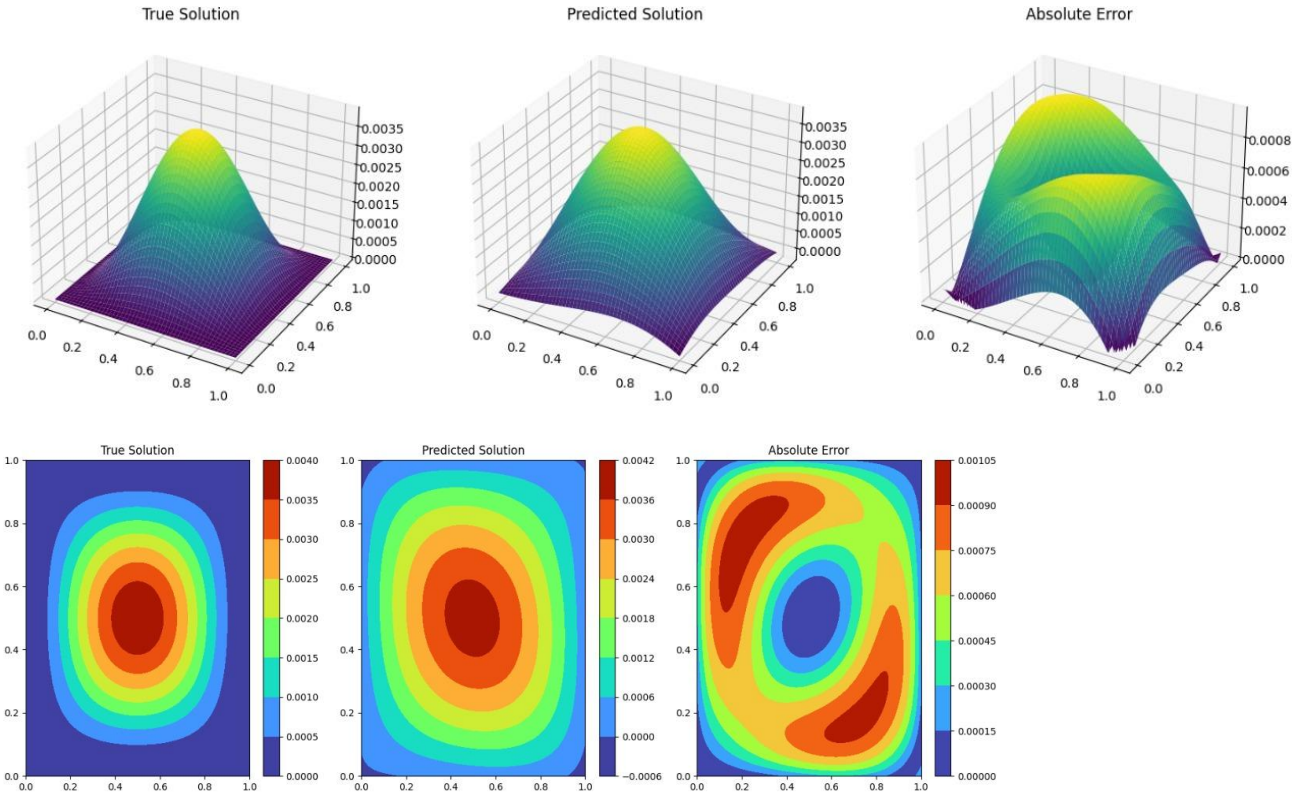
H1 Relative: 0.09169480840086616

H2 Relative: 0.11836172882203878

Loss Curve of 3.1



Results Example 3.2



Epoch 6000

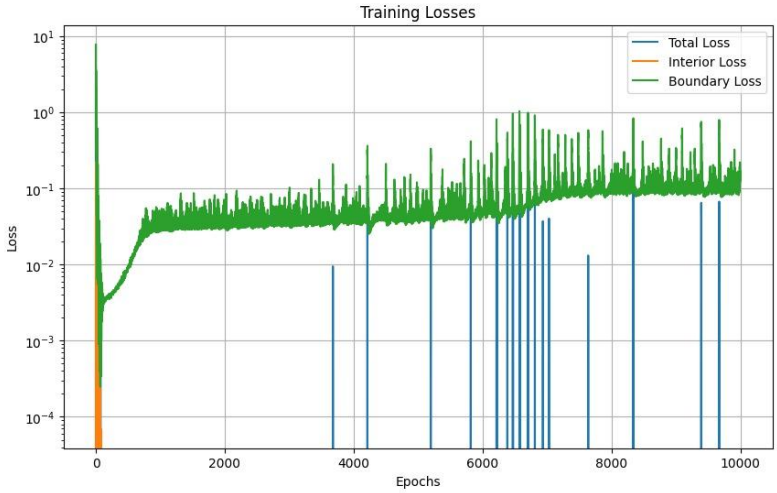
L2 Error: 0.0005970842321403325

L2 Relative Error: 0.376539633146755

H1 Relative Error: 0.505611430441448

H2 Relative Error: 0.8424360131116138

Loss Curve of 3.2



Contributions

Every One in team has equally Contributed to this Project Completion.