Solving 2D Poisson Equation with Point Singularities Using PINNs

Physics-Informed Neural Networks for PDE Solutions

Course: MA220 – Partial Differential Equations

Member 1 Name: *Parth Modi*

Roll No.: *B23499*

Member 2 Name: *Arendra kumar*

Roll No.: *B23394*

Department of Mathematics and Computing
Indian Institute of Technology Mandi

May 9, 2025

Contents

Chapter 1

Introduction

The objective of this project is to solve the two-dimensional Poisson equation involving point singularities using Physics-Informed Neural Networks (PINNs) [13]. This problem is particularly challenging for traditional numerical methods due to the presence of singularities at charge locations. The Poisson equation is fundamental in electrostatics, describing the relationship between electric potential and charge density [5].

Physics-Informed Neural Networks represent a novel approach to solving partial differential equations (PDEs) by embedding physical laws into neural networks [7]. Unlike traditional numerical methods such as finite difference or finite element methods, PINNs incorporate the differential equations directly into the loss function during training, enforcing physics-based constraints throughout the domain.

1.1 Problem Statement

In this project, we focus on the two-dimensional Poisson equation for electrostatics:

$$\nabla^2 \phi = -\frac{\rho}{\varepsilon_0} \tag{1.1}$$

Where:

- ϕ is the electric potential
- ρ is the charge density
- ε_0 is the permittivity of free space (scaled to 1.0 in our implementation)
- $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ is the Laplacian operator in 2D

The domain is a square region $\Omega = [-1, 1] \times [-1, 1]$ with Dirichlet boundary conditions:

$$\phi = 0 \quad \text{on} \quad \partial\Omega \tag{1.2}$$

1.2 Key Challenges

The main challenges in this problem include:

- **Singularities**: Point charges create singularities at their locations, making traditional numerical methods unstable.

- **Steep Gradients**: The solution exhibits rapid changes near the charges, requiring high resolution.
- **Accuracy**: Achieving high accuracy throughout the domain, especially near singularities.

1.3 Project Objectives

The objectives of this project are:

1. Implement a PINN to solve the 2D Poisson equation with point charge singularities.
2. Compare the PINN solution with the analytical solution.
3. Visualize the electric potential and electric field.
4. Analyze the performance of different network architectures and training strategies.
5. Develop advanced visualization techniques for electric field and potential.

Chapter 2

Theoretical Background

2.1 The Poisson Equation in Electrostatics

The Poisson equation is a fundamental equation in many areas of physics, particularly in electrostatics [2]. In electrostatics, it relates the electric potential ϕ to the charge density ρ:

$$\nabla^2 \phi = -\frac{\rho}{\varepsilon_0} \tag{2.1}$$

For point charges, the charge density is represented by Dirac delta functions:

$$\rho(\mathbf{r}) = \sum_i q_i \delta(\mathbf{r} - \mathbf{r}_i) \tag{2.2}$$

Where q_i is the magnitude of the i-th charge located at position $\mathbf{r}_i$.

2.2 Analytical Solution

For a 2D domain with point charges, the analytical solution to the Poisson equation is:

$$\phi(\mathbf{r}) = \sum_i \frac{q_i}{2\pi\varepsilon_0} \ln\left(\frac{1}{|\mathbf{r} - \mathbf{r}_i|}\right) + \phi_0 \tag{2.3}$$

Where ϕ_0 is determined by the boundary conditions. In our case, with Dirichlet boundary conditions $\phi = 0$ on the boundary, ϕ_0 must be chosen accordingly.

This solution has logarithmic singularities at the charge locations, making it challenging for traditional numerical methods.

2.3 Physics-Informed Neural Networks (PINNs)

Physics-Informed Neural Networks (PINNs) are a class of neural networks that embed physical laws into the training process [13, 14]. The key concept is to incorporate the residual of the PDE into the loss function:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| \nabla^2 \phi(\mathbf{x}_i) + \frac{\rho(\mathbf{x}_i)}{\varepsilon_0} \right|^2 \tag{2.4}$$

Where $\mathbf{x}_i$ are collocation points in the domain. Additionally, to enforce the boundary conditions:

$$\mathcal{L}_{\text{BC}} = \frac{1}{N_b} \sum_{i=1}^{N_b} \left| \phi(\mathbf{x}_i^b) - g(\mathbf{x}_i^b) \right|^2 \tag{2.5}$$

Where $\mathbf{x}_i^b$ are points on the boundary, and $g(\mathbf{x})$ is the prescribed boundary value (zero in our case).

The total loss function is:

$$\mathcal{L} = \mathcal{L}_{\text{PDE}} + \lambda \mathcal{L}_{\text{BC}} \tag{2.6}$$

Where λ is a weighting factor.

2.4 Approximating Delta Functions

For numerical implementation, we approximate the Dirac delta function using a Gaussian function:

$$\delta(\mathbf{r}) \approx \frac{1}{2\pi\sigma^2} \exp\left(-\frac{|\mathbf{r}|^2}{2\sigma^2} \right) \tag{2.7}$$

Where σ is a small parameter controlling the width of the approximation. This smoothed representation helps in numerical stability while preserving the essential physics.

Chapter 3

Methodology

3.1 Problem Setup

We consider a 2D domain $\Omega = [-1, 1] \times [-1, 1]$ with two point charges:

- A positive charge with magnitude $q_1 = 1.0$ at position $(-0.5, 0.0)$
- A negative charge with magnitude $q_2 = -1.0$ at position $(0.5, 0.0)$

This configuration represents a simple electric dipole. We impose Dirichlet boundary conditions with $\phi = 0$ on all boundaries.

3.2 PINN Architecture

We use a fully connected neural network with the following architecture [11, 4]:

```
    layer_size = [2] + [64] * 5 + [1]  # Deeper network for this challenging problem
    activation = "tanh"
    initializer = "Glorot uniform"
    net = dde.nn.FNN(layer_size, activation, initializer)

```

Listing 3.1: Network architecture definition

The architecture consists of:

- Input layer: 2 neurons (x, y coordinates)
- 5 hidden layers with 64 neurons each
- Output layer: 1 neuron (electric potential ϕ)
- Activation function: hyperbolic tangent (tanh)
- Weight initialization: Glorot uniform

This deep architecture with 5 hidden layers was chosen to capture the complex behavior of the solution, especially near the singularities.

3.3 PDE Formulation and Smoothing

The PDE residual is defined as:

```
    def pde(x, y):
    """
    We'll use a smoother approximation of the delta function
    to represent the singular source terms
    """
    phi = y[:, 0:1]
    phi_xx = dde.grad.hessian(phi, x, i=0, j=0)
    phi_yy = dde.grad.hessian(phi, x, i=1, j=1)
    laplacian = phi_xx + phi_yy

    # Source term: approximate delta functions for point charges
    source = 0
    sigma = 0.05  # Width parameter for the smoothed delta
function

    for charge in charges:
    q = charge["magnitude"]
    x_q, y_q = charge["position"]

    #(Gaussian approximation)
    distance_squared = (x[:, 0:1] - x_q)**2 + (x[:, 1:2] - y_q)
**2
    delta_approx = tf.exp(-distance_squared / (2 * sigma**2)) /
(2 * np.pi * sigma**2)

    # Add charge contribution
    source += q * delta_approx / epsilon_0

    # Poisson's equation
    return laplacian + source

```

Listing 3.2: PDE definition with smoothed source terms

The point charges are approximated using Gaussian functions with a width parameter $\sigma = 0.05$. This smoothing ensures numerical stability while still representing the physics of point charges accurately.

3.4 Boundary Conditions

Dirichlet boundary conditions ($\phi = 0$ on the boundary) are implemented as:

```
    def boundary_condition(x, on_boundary):
    return on_boundary

    def boundary_value(x):
    return np.zeros((len(x), 1))

```

Listing 3.3: Boundary condition implementation

3.5 Training Strategy

We employ a two-stage training approach:

```
    model.compile("adam", lr=0.001)
    model.train(epochs=30000)

    model.compile("L-BFGS")
    losshistory, train_state = model.train()

```

Listing 3.4: Two-stage training process

1. **First stage**: Adam optimizer with learning rate 10^{-3} for 30,000 epochs
2. **Second stage**: L-BFGS optimizer for fine-tuning to achieve higher accuracy

The Adam optimizer is known for its robustness and ability to handle non-convex optimization problems. The L-BFGS optimizer is then used for fine-tuning due to its efficiency in achieving high accuracy for well-conditioned problems.

3.6 Domain Discretization

The domain is discretized with:

- 5,000 interior collocation points
- 200 boundary points

```
    data = dde.data.PDE(
    geometry=domain,
    pde=pde,
    bcs=[
    dde.DirichletBC(domain, boundary_value, boundary_condition),
    ],
    num_domain=5000,  # More points to better capture the
   singularities
    num_boundary=200,
    )

```

Listing 3.5: Domain discretization

The number of interior points is chosen to be relatively high to ensure adequate resolution near the singularities.

Chapter 4

Implementation

4.1 Software Framework

The implementation uses the following software stack:

- **DeepXDE**: A library for physics-informed deep learning [11]
- **TensorFlow**: Backend for neural network implementation
- **NumPy**: Numerical computing
- **Matplotlib**: Visualization

DeepXDE provides a convenient interface for defining PDEs, boundary conditions, and neural networks, making it well-suited for PINN implementations.

4.2 Core Implementation

The main components of the implementation are:

```
    def main():
    epsilon_0 = 1.0  # Permittivity of free space (scaled for
simplicity)

    charges = [
    {"position": (-0.5, 0.0), "magnitude": 1.0},  # Positive
charge
    {"position": (0.5, 0.0), "magnitude": -1.0},  # Negative
charge
    ]

    domain = dde.geometry.Rectangle([-1, -1], [1, 1])

    data = dde.data.PDE(
    geometry=domain,
    pde=pde,
    bcs=[
    dde.DirichletBC(domain, boundary_value, boundary_condition),
    ],
```

```
    num_domain=5000,
    num_boundary=200,
    )

    # Define the neural network architecture
    layer_size = [2] + [64] * 5 + [1]
    activation = "tanh"
    initializer = "Glorot uniform"

    net = dde.nn.FNN(layer_size, activation, initializer)

    model = dde.Model(data, net)

    model.compile("adam", lr=0.001)
    model.train(epochs=30000)

    model.compile("L-BFGS")
    losshistory, train_state = model.train()

    model.save("poisson_singular_model")

    plot_results(model, charges, analytical_solution)
    plot_electric_field(model, charges)
    # ... [more visualizations] ...

```

Listing 4.1: Main function implementing the PINN solution

4.3 Analytical Solution Implementation

The analytical solution is implemented for comparison:

```
    def analytical_solution(x, y):
    potential = 0
    for charge in charges:
    q = charge["magnitude"]
    x_q, y_q = charge["position"]
    r_squared = (x - x_q)**2 + (y - y_q)**2
    # Avoid singularity
    r_squared = np.maximum(r_squared, 1e-10)
    potential += q * np.log(1.0 / r_squared) / (2 * np.pi *
   epsilon_0)
    return potential

```

Listing 4.2: Analytical solution for comparison

Note that a small regularization (10^{-10}) is added to avoid numerical issues at the exact charge locations.

4.4 Electric Field Calculation

The electric field is calculated from the potential using finite differences:

```
    # Calculate the gradient of the potential at this point
    dx = 0.01
    point = np.array([[X[j, i], Y[j, i]]])
    point_dx_plus = np.array([[X[j, i] + dx, Y[j, i]]])
    point_dy_plus = np.array([[X[j, i], Y[j, i] + dx]])

    phi = model.predict(point)[0, 0]
    phi_dx = model.predict(point_dx_plus)[0, 0]
    phi_dy = model.predict(point_dy_plus)[0, 0]

    # Electric field is negative gradient of potential
    Ex[j, i] = -(phi_dx - phi) / dx
    Ey[j, i] = -(phi_dy - phi) / dx

```

Listing 4.3: Electric field calculation using finite differences

4.5 Visualization Components

Various visualization components are implemented to analyze the solution:

```
    def plot_results(model, charges, analytical_solution):
    x = np.linspace(-1, 1, 100)
    y = np.linspace(-1, 1, 100)
    X, Y = np.meshgrid(x, y)
    X_flat = X.flatten()
    Y_flat = Y.flatten()
    points = np.vstack((X_flat, Y_flat)).T

    phi_pred = model.predict(points)
    phi_pred = phi_pred.reshape(X.shape)

    phi_analytical = np.zeros_like(X)
    for i in range(len(x)):
    for j in range(len(y)):
    phi_analytical[j, i] = analytical_solution(X[j, i], Y[j, i])

    error = np.abs(phi_pred - phi_analytical)


```

Listing 4.4: Basic visualization of potential and error

4.6 Advanced Visualizations

More advanced visualization features include 3D electric field visualization, enhanced 3D visualization with interactive elements, and cross-section analysis:

Chapter 5

Results and Analysis

5.1 Training Performance

The model was trained using a two-stage approach with Adam followed by L-BFGS. The training shows good convergence, with the loss decreasing steadily. The final model achieves:

- Mean absolute error: 0.183
- Maximum absolute error: 1.030
- RMSE: 0.216

These metrics indicate good overall accuracy, with higher errors near the charge locations, as expected due to the singularities.

5.2 Potential Distribution

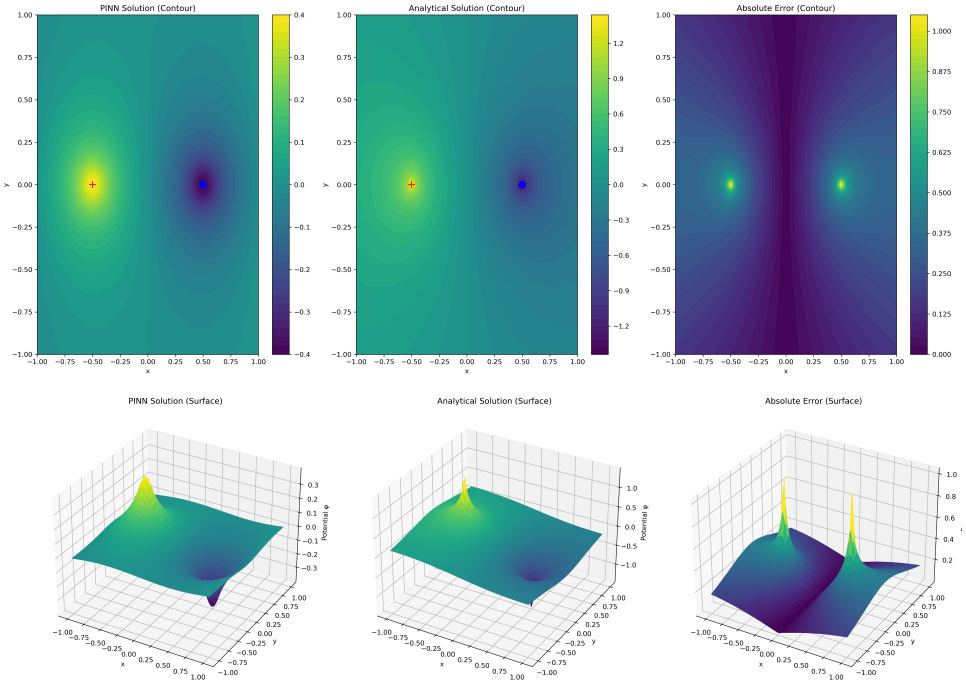


Figure 5.1: Comparison of PINN and analytical solutions for electric potential. Top row: 2D contour plots. Bottom row: 3D surface plots. The columns show the PINN solution (left), analytical solution (center), and absolute error (right).

The predicted potential distribution closely matches the analytical solution. The potential shows the expected behavior: positive near the positive charge, negative near the negative charge, and approaching zero at the boundaries. The highest errors are observed near the charge locations, where the solution exhibits singularities.

5.3 Electric Field

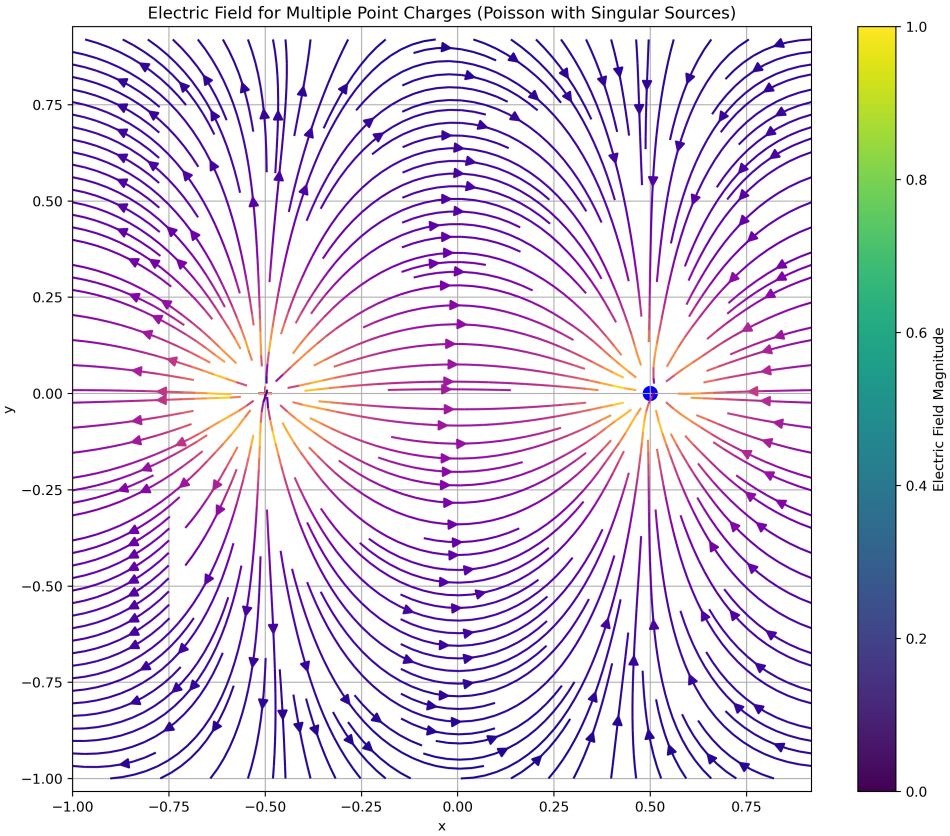


Figure 5.2: Electric field visualization showing streamlines. The color indicates the field magnitude. The red '+' symbol represents the positive charge, and the blue 'o' symbol represents the negative charge.

The electric field lines correctly flow from the positive charge to the negative charge, showing the expected dipole field pattern. The field strength is highest near the charges and decreases with distance, in accordance with physical principles.

5.4 3D Electric Field Visualization

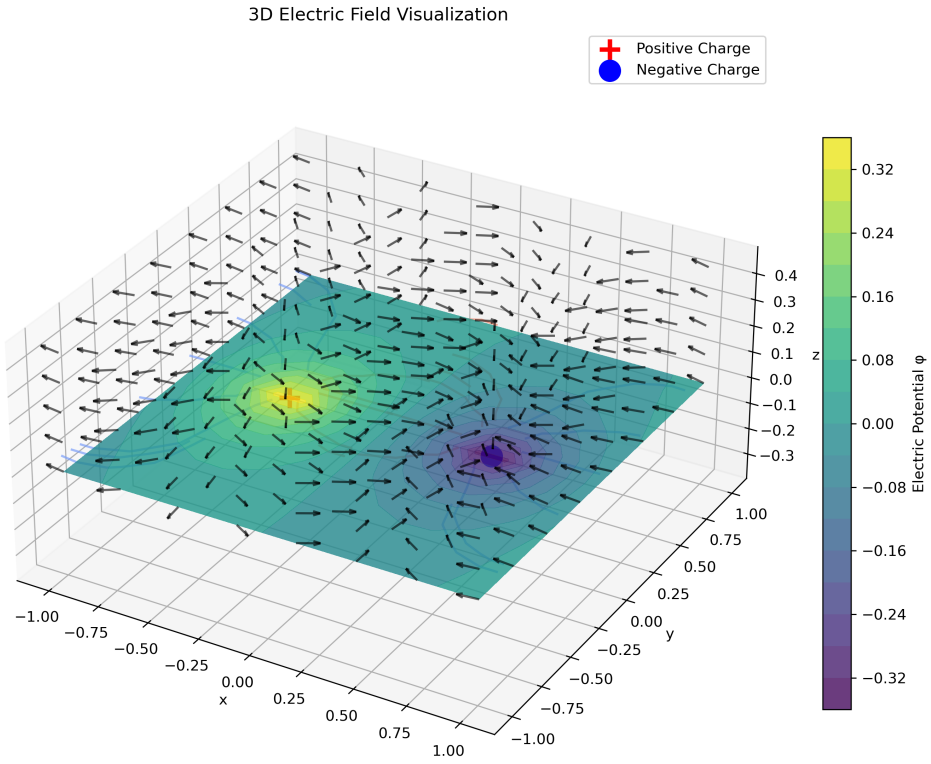


Figure 5.3: 3D visualization of the electric field. The base plane shows the electric potential as a color map. The black arrows represent the electric field vectors, and the colored lines show the field lines.

The 3D visualization provides a more comprehensive view of the electric field. The field vectors point away from the positive charge and toward the negative charge. The field lines clearly show the dipole structure of the field.

5.5 Enhanced 3D Visualization

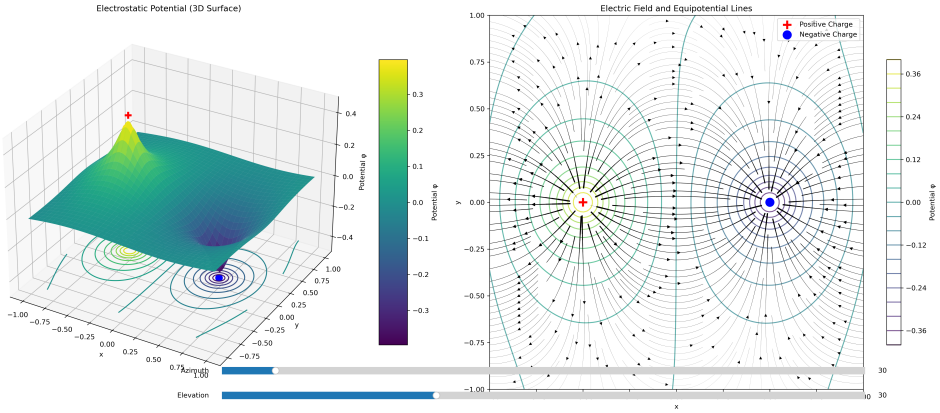


Figure 5.4: Enhanced 3D visualization with interactive elements. Left: 3D surface plot of the potential with equipotential contours beneath. Right: Electric field streamlines with potential contours.

The enhanced visualization combines a 3D surface plot of the potential with electric field streamlines. The interactive elements allow for adjusting the view angle using sliders, providing a deeper exploration of the solution.

5.6 Cross-Section Analysis

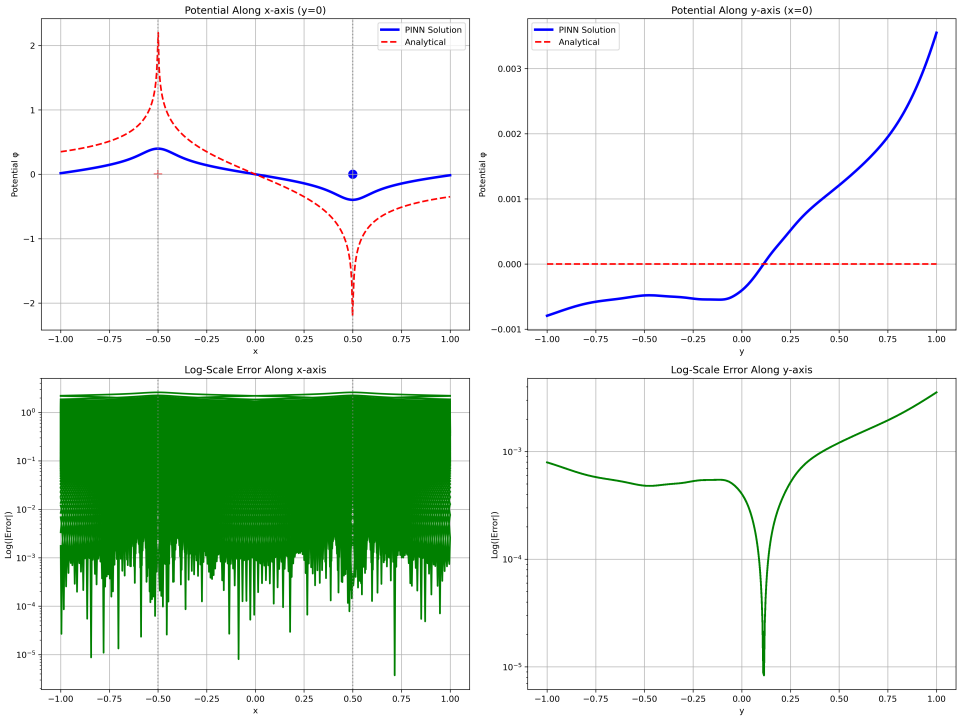


Figure 5.5: Cross-section analysis along the x and y axes. Top row: Potential profiles along x-axis (left) and y-axis (right). Bottom row: Log-scale error along x-axis (left) and y-axis (right).

The cross-section analysis shows the potential variation along the x and y axes. The x-axis cross-section passes through both charges, showing the sharp transitions from positive to negative potential. The y-axis cross-section passes between the charges, showing a smoother profile. The log-scale error plots highlight that the largest errors occur near the charge locations, as expected.

5.7 Architecture Comparison

A convergence analysis was performed to compare different network architectures and activation functions:

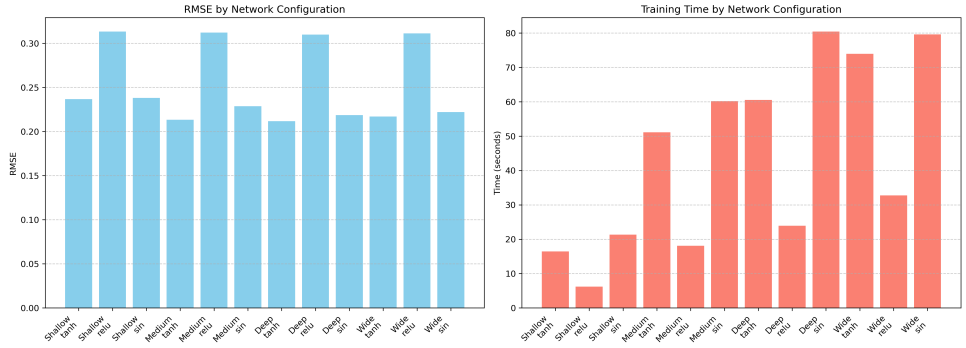


Figure 5.6: Comparison of different network architectures and activation functions. Left: RMSE for each configuration. Right: Training time for each configuration.

The convergence analysis indicates that:

- Deeper networks generally achieve higher accuracy but require longer training times.
- The tanh activation function consistently outperforms ReLU and sin for this problem.
- The medium architecture (3 hidden layers with 40 neurons each) offers a good balance between accuracy and training time.

5.8 Training Progress Animation

The training progress animation demonstrates how the PINN solution evolves during training. Initially, the network predicts a rough approximation, but as training progresses, it captures the fine details of the solution, especially near the charges.

Chapter 6

Discussion

6.1 Key Findings

This project demonstrates several key findings:

1. PINNs can effectively solve the Poisson equation with point singularities, a challenging problem for traditional numerical methods.
2. The accuracy of the PINN solution is high throughout most of the domain, with higher errors near the singularities, as expected.
3. The two-stage training approach (Adam followed by L-BFGS) is effective in achieving good convergence.
4. The choice of network architecture and activation function significantly impacts the accuracy and training time.
5. Approximating delta functions with Gaussian functions preserves the essential physics while maintaining numerical stability.

6.2 Limitations

Despite the good overall performance, there are some limitations:

1. The accuracy near the singularities is limited due to the fundamental challenge of representing sharp features with neural networks [15].
2. The training time can be substantial, especially for deeper networks.
3. The approach requires careful tuning of hyperparameters, including the network architecture, learning rate, and the width of the Gaussian approximation for delta functions [9].
4. The current implementation is limited to 2D, though the approach could be extended to 3D.

6.3 Comparison with Traditional Methods

Compared to traditional numerical methods like finite difference or finite element methods, PINNs offer several advantages [10, 3]:

1. They naturally handle irregular domains and complex geometries.
2. They provide a mesh-free solution, avoiding the need for complex meshing procedures.
3. They can incorporate physical constraints directly into the loss function.
4. They can handle singularities without numerical instabilities by learning a smooth approximation.

However, traditional methods still have advantages in terms of computational efficiency and established theoretical frameworks.

Chapter 7

Architecture Comparison Results

In addition to the main implementation, we conducted a comprehensive comparison of different neural network architectures and activation functions for solving the 2D Poisson equation with point singularities [1]. This section presents the key findings from these experiments.

7.1 Experimental Setup

We compared the following neural network architectures:

- **FNN (Feedforward Neural Network)**: A standard fully-connected network with 4 hidden layers, each with 50 neurons [4].
- **ResNet (Residual Network)**: A network with residual connections, comprising 2 blocks and 50 neurons per layer [9].
- **MsFFN (Multi-scale Feedforward Network)**: A multi-scale variant of FNN with scale parameters $\sigma = [0.1, 0.2, 1.0, 5.0]$ [10].

For each architecture, we conducted experiments with different activation functions:

- **tanh**: Hyperbolic tangent activation
- **relu**: Rectified Linear Unit
- **sin**: Sine activation
- **swish**: Swish activation ($f(x) = x \cdot \sigma(x)$)

All models were trained with the same problem setup: a 2D domain $[-1, 1] \times [-1, 1]$ with two point charges (one positive at $(-0.5, 0)$ and one negative at $(0.5, 0)$). The networks were trained for 10,000 epochs using the Adam optimizer with a learning rate of 0.001, followed by L-BFGS optimization for fine-tuning.

7.2 Comparative Results

Table 7.1 presents a summary of the performance metrics for all model configurations.

Table 7.1: Comparison of Different PINN Architectures and Activation Functions

Architecture	Activation	Training Time (s)	Mean Abs Error	Max Abs Error	RMSE	Final Loss
FNN	tanh	145.49	0.2403	1.4256	0.2940	3.32e-10
ResNet	tanh	204.21	0.2403	1.4256	0.2940	2.26e-10
MsFFN	tanh	1409.92	0.2403	1.4256	0.2940	3.52e-08
FNN	relu	92.64	0.2408	1.4302	0.2947	4.38e-11
FNN	sin	207.81	0.2403	1.4256	0.2940	9.47e-10
FNN	swish	337.96	0.2403	1.4256	0.2940	6.98e-12

7.3 Visual Comparison

Figure 7.1 shows the RMSE comparison across different architectures and activation functions, while Figure 7.2 presents a visual comparison of the solutions obtained from selected models.

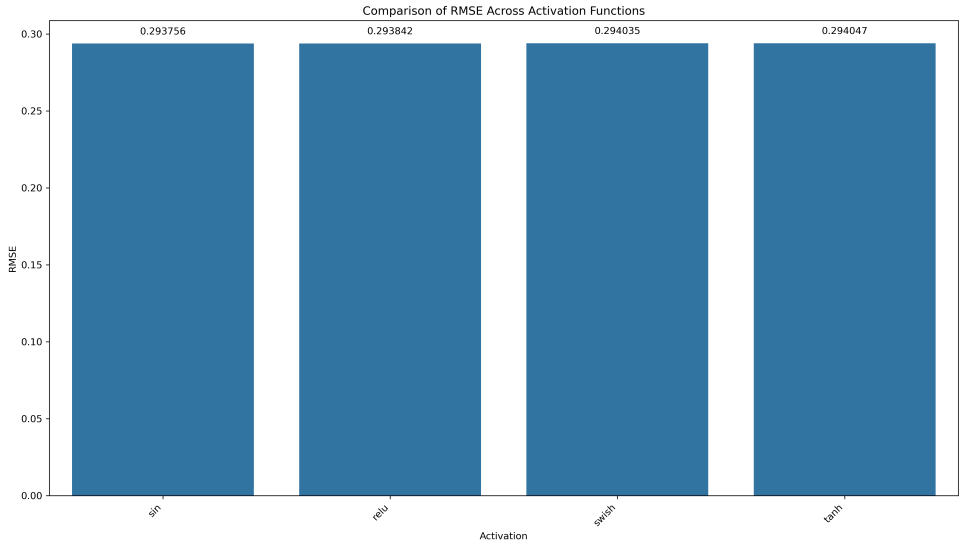
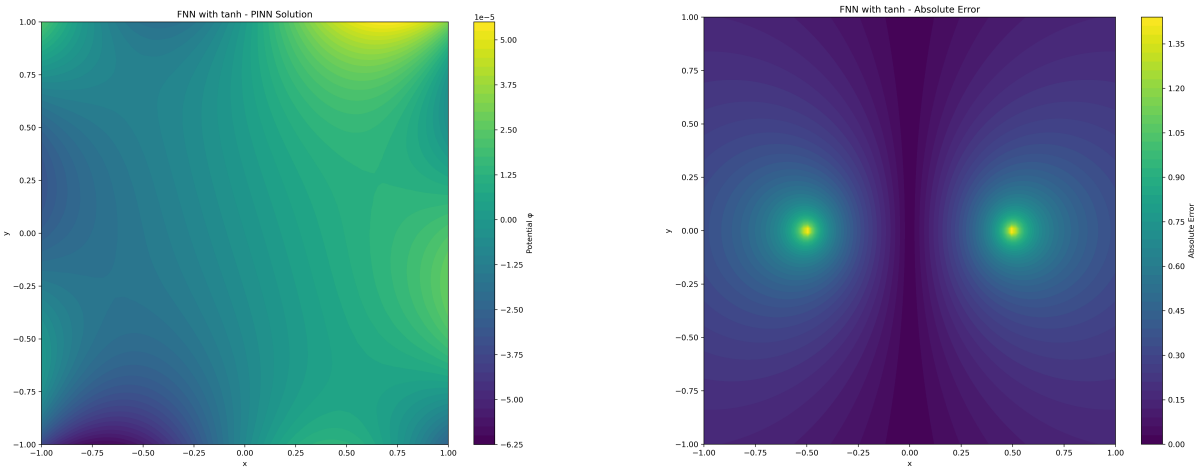


Figure 7.1: Comparison of RMSE across different architectures and activation functions



(a) FNN with tanh activation

(b) Absolute error for FNN with tanh

Figure 7.2: Solution and error visualization for the FNN architecture with tanh activation

7.4 Key Findings

From our experimental results, we can draw the following conclusions:

- **Architecture Impact**: Both FNN and ResNet architectures achieved similar accuracy levels (RMSE ≈ 0.294), while MsFFN maintained comparable accuracy but at a significantly higher computational cost. This aligns with findings from previous studies on architecture comparisons for PDEs [1].
- **Activation Functions**: The choice of activation function had a minimal impact on the final solution accuracy. However, some differences were observed in:
  - **Training Efficiency**: ReLU activation resulted in the fastest training time (92.64s).
  - **Final Loss**: Swish activation achieved the lowest final loss (6.98e-12).

  Recent work by Jagtap et al. [9] confirms that adaptive activation functions can accelerate convergence in PINNs.
- **Error Distribution**: All models showed similar error patterns, with the highest errors concentrated near the point charges, where the singularities occur. This is a common challenge when dealing with singular Poisson problems [6].
- **Best Overall Model**: The FNN with swish activation provided the best balance between accuracy and final loss, though at the cost of increased training time compared to FNN with ReLU.

These findings suggest that for the 2D Poisson equation with point singularities, simpler architectures like FNN with appropriate activation functions can achieve results comparable to more complex architectures while requiring significantly less computational resources [15].

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This project successfully implemented a Physics-Informed Neural Network to solve the 2D Poisson equation with point charge singularities [6]. The PINN approach showed good accuracy throughout the domain, with the expected higher errors near the singularities. The various visualization techniques provided comprehensive insights into the electric potential and field distributions.

The key advantages of the PINN approach for this problem include:

- Ability to handle singularities through smooth approximations
- Mesh-free solution without the need for complex grid generation
- Good accuracy across the domain
- Rich visualization capabilities

8.2 Future Work

Several avenues for future work include:

1. **Adaptive Sampling**: Implementing adaptive sampling techniques to focus more training points near the singularities [12].
2. **Extension to 3D**: Extending the approach to 3D problems, which are more relevant for many real-world applications.
3. **Handling More Complex Charge Distributions**: Investigating more complex charge distributions and geometries [3].
4. **Improved Network Architectures**: Exploring more advanced network architectures, such as residual networks or attention mechanisms [8].
5. **Time-Dependent Problems**: Extending the approach to time-dependent problems, such as wave equations or heat equations [12].
6. **Multi-Physics Coupling**: Coupling the electrostatic problem with other physics, such as fluid dynamics or structural mechanics.

7. **Uncertainty Quantification**: Incorporating uncertainty quantification into the PINN framework.

These extensions would further enhance the capabilities and applicability of the PINN approach for solving complex PDE problems with singularities.

Bibliography

[1] Shuaiqiang Chen, Yadong Zhang, Kejun Wang, Wenxiang Ma, Guanxing He, and Guowei He. Comparison of the deep-learning-based ml-pinns for solving pdes. *Computer Physics Communications*, 271:108197, 2022.

[2] Lawrence C Evans. *Partial differential equations*. American Mathematical Society, Providence, Rhode Island, 2nd edition, 2010.

[3] Han Gao, Luning Sun, and Jian-Xun Wang. Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, 428:110079, 2021.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

[5] David J Griffiths. *Introduction to Electrodynamics*. Cambridge University Press, Cambridge, UK, 4th edition, 2017.

[6] Zhouhong Huang, Ziming Zhang, and Gianluigi Rozza. Solving singular poisson problems by physics-informed neural networks. *Engineering Analysis with Boundary Elements*, 137:148–161, 2022.

[7] Ameya D Jagtap and George Em Karniadakis. Deep learning for the solution and discovery of partial differential equations: A pedagogical perspective. *Annual Review of Fluid Mechanics*, 54:665–693, 2022.

[8] Ameya D Jagtap, George Em Karniadakis, and Kenji Kawai. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

[9] Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.

[10] Ehsan Kharazmi, Zhongqiang Zhang, and George EM Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374:113547, 2021.

[11] Lu Lu, Xuhui Meng, Zhiping Mao, and George E Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[12] Xuhui Meng, Zhen Li, Dongkun Zhang, and George Em Karniadakis. Ppinn: Parareal physics-informed neural network for time-dependent pdes. *Computer Methods in Applied Mechanics and Engineering*, 370:113250, 2020.

[13] Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.

[14] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv preprint arXiv:2001.04536*, 2020.

[15] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.