HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

SIMULATION OF 2-DIMENSIONAL HEAT CONDUCTION WITH PHYSICS-INFORMED NEURAL NETWORKS

By Samman Aryal
Matriculation Nr.: 34867

A thesis submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in
Electrical Engineering and Information Technology

Supervisors
Prof. Dr.-Ing. Lothar Berger
M. Sc. Stephan Scholz

Hochschule Ravensburg Weingarten University of Applied Sciences
April 2024

Abstract

Heat conduction is a fundamental process that affects countless aspects of our daily lives, from how we heat our homes to the efficiency of industrial systems. In this thesis, I explore an innovative approach to simulating two-dimensional heat conduction using Physics-Informed Neural Networks (PINNs).I begin by laying the groundwork in heat conduction principles, explaining the key concepts and the importance of boundary conditions which will be relevant to this thesis. As I delve into how neural networks can solve differential equations, I highlight the training process and techniques that improve model performance, making these advanced concepts approachable.

The heart of this work is a PINN framework that incorporates the laws of physics directly into the neural network design. This ensures that the simulations adhere closely to the governing equations of heat conduction. Through engaging case studies, I evaluate the effectiveness of this approach, testing it under various heat sources and boundary conditions.The results are promising: not only does this method accurately capture heat distribution, but it also emerges as a strong alternative to traditional techniques like the Finite Difference Method (FDM) and Finite Element Method (FEM). These findings underscore the potential of PINNs to enhance computational efficiency and flexibility, opening doors for future research in more complex thermal systems.In summary, this thesis adds to the growing dialogue between machine learning and engineering, offering valuable insights into how we can harness the power of PINNs to tackle practical heat conduction challenges. I hope this work inspires further exploration in this exciting intersection of fields.

Contents

List of Figures

List of Tables

1 Introduction

The study of heat transfer phenomena plays a crucial role in various engineering and scientific disciplines, including thermal management, material design, and energy systems. Among the different modes of heat transfer, conduction is a fundamental process that governs the transfer of thermal energy within a medium or between multiple media in direct contact. Understanding and accurately modeling heat conduction is essential for optimizing thermal performance, ensuring safety, and improving energy efficiency in numerous applications.Traditional numerical methods, such as the finite element method (FEM) and the finite difference method (FDM), have been widely employed to solve heat conduction problems. However, these methods often require extensive computational resources, particularly for complex geometries or time-dependent problems. Additionally, they rely on discretization techniques that can introduce numerical errors and stability issues, which may compromise the accuracy of the solutions.

In recent years, the field of machine learning has witnessed remarkable advancements, with neural networks emerging as powerful tools for solving complex problems across various domains. Physics-Informed Neural Networks (PINNs) represent a novel approach that combines the flexibility of neural networks with the principles of physics-based modeling. By incorporating governing equations and boundary conditions directly into the neural network architecture, PINNs can learn the underlying physical laws and provide accurate solutions without extensive discretization or mesh generation. This thesis focuses on the simulation of two-dimensional (2D) heat conduction problems using PINNs, aiming to develop a robust PINN-based framework to simulate 2D heat conduction with Neumann boundary conditions, thereby leveraging the capabilities of neural networks to integrate physical laws into the learning process.

The study will explore various heat distribution scenarios, including a plate with a localized hot region, a square plate with a constant central heating element, heat redistribution, and the natural cooling of a hot plate. To validate the PINN model, its solutions will be compared with analytical solutions and traditional numerical methods to evaluate accuracy and efficiency. Additionally, the computational performance of the PINN model will be assessed in terms of speed, resource usage, and scalability. Finally, the research aims to identify potential applications of the developed PINN model in real-world scenarios, such as thermal management systems and material design, highlighting its relevance and impact in engineering and scientific domains.

The successful implementation of PINNs for simulating 2D heat conduction problems can potentially revolutionize the way thermal analysis is performed, offering a more efficient and accurate approach compared to traditional methods. By leveraging the power of neural networks and incorporating physical laws, PINNs have the potential to provide accurate solutions while reducing computational complexity and enabling real-time simulations.This thesis is structured to provide a comprehensive understanding of the research topic, covering theoretical background, problem definition, methodology, simulation results, discussion, and conclusions. It aims to contribute to the growing field of Physics-Informed Neural Networks and their applications in solving partial differential equations, with a specific focus on heat conduction problems.

2 Theoretical Background

This section introduces the fundamental concepts and theories that underpin the research on simulating two-dimensional heat conduction using Physics-Informed Neural Networks (PINNs). It covers the principles of heat conduction, the mathematical formulation of the problem, and an overview of neural networks and their application in solving partial differential equations (PDEs). By embedding the physics directly into the neural network architecture, PINNs offer a novel approach to modeling complex thermal processes, leveraging both data-driven and physics-based strategies to achieve accurate predictions.

2.1 Introduction to Heat Conduction

Heat conduction is a fundamental process of heat transfer, where energy is transferred from a region of higher temperature to a region of lower temperature within a medium or between different media in direct physical contact.It is one of the three basic modes of thermal energy transport (convection and radiation being the other two) and is involved in virtually all process heat-transfer operations.

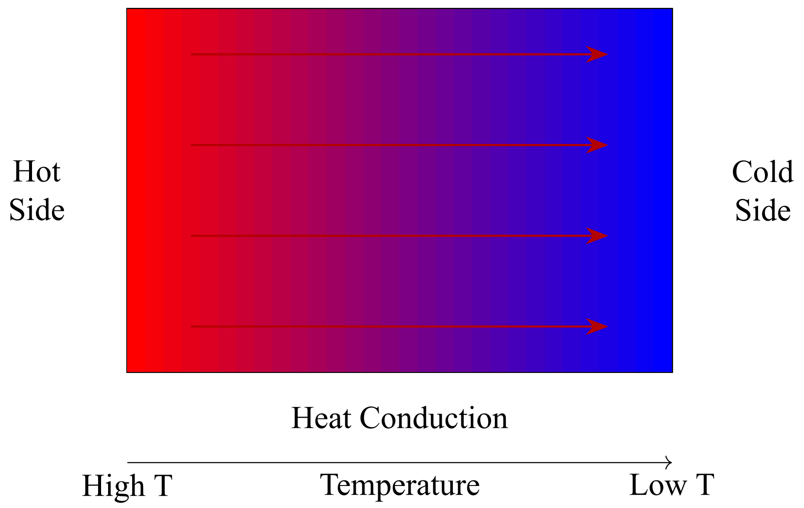


Figure 2.1: General Heat Conduction Process

This process occurs at the molecular level, where kinetic energy is transferred through collisions between neighboring particles. Heat conduction is an important phenomenon in various fields, including engineering, physics, and everyday life. It plays a crucial role in designing efficient thermal management systems, understanding natural processes, and improving energy efficiency across numerous applications.[1]

2.1.1 Fundamental Principles of Heat Conduction

The rate of heat conduction depends on several factors, including the temperature gradient, the material properties, and the geometry of the medium. These factors collectively determine how effectively heat is transferred through a substance, influencing both the speed and efficiency of the process.

The basic law governing heat conduction is Fourier's law, which states that the heat flux (rate of heat transfer per unit area) is proportional to the negative temperature gradient that is

$$\vec{q} = -k\nabla T$$

where $\vec{q}$ is the heat flux vector, k is the thermal conductivity of the material (a measure of its ability to conduct heat), and ∇T is the temperature gradient. Fourier's law provides a fundamental framework for analyzing and predicting heat transfer in a wide array of applications, from industrial processes to environmental phenomena.[2]

2.1.1.1 Thermal Conductivity and Material Properties

The thermal conductivity of a material is a crucial property that determines its ability to conduct heat. Thermal conductivity is influenced by the material's atomic structure and bonding, as these factors dictate how easily energy can be transferred between particles. Materials with high thermal conductivity, such as metals, are good heat conductors, while materials with low thermal conductivity, such as insulators, are poor heat conductors. The thermal conductivity can be influenced by various factors, including the material's composition, density, and temperature.

Material Composition: Different materials exhibit varying levels of thermal conductivity. For instance, metals such as copper and aluminum have high thermal conductivities due to their free electrons, which facilitate efficient energy transfer. On the other hand, materials like wood, rubber, and certain plastics have low thermal conductivities and act as insulators, trapping heat and slowing its transfer.

Density: The density of a material can also affect its thermal conductivity. Denser materials generally have higher thermal conductivities because the particles are closely packed, facilitating energy transfer between them. However, this relationship is not always straightforward, as other factors such as material structure and bonding also play significant roles.

Temperature Dependency: The thermal conductivity of a material can change with temperature. For example, the thermal conductivity of metals typically decreases with increasing temperature due to increased lattice vibrations, which scatter heat-carrying electrons. Conversely, the thermal conductivity of non-metals may increase with temperature as increased molecular motion enhances energy transfer.[3]

2.1.1.2 Heat Conduction in Combination with Other Mechanisms

Heat conduction is often accompanied by other heat transfer mechanisms, such as convection and radiation. These mechanisms interact with conduction to influence the overall heat transfer process, particularly in complex systems where multiple phases and materials are involved. Convection involves the transfer of heat through the motion of fluids, while radiation is the transfer of energy through electromagnetic waves. In many practical applications, heat transfer occurs through a combination of these mechanisms, and it is essential to consider their relative contributions to the overall heat transfer process. [2]

Convection: This mechanism involves the bulk movement of fluid (liquid or gas) carrying heat from one place to another. It can be natural (driven by buoyancy forces due to density differences) or forced (driven by external means like fans or pumps). Convection plays a significant role in heat transfer in fluids and is often coupled with conduction at the boundary layer where the fluid contacts a solid surface. Understanding convection is crucial for applications involving fluid flow, such as HVAC systems, industrial cooling, and atmospheric processes.

Radiation: Unlike conduction and convection, radiation does not require a medium and can occur in a vacuum. It involves the transfer of energy through electromagnetic waves, primarily in the infrared spectrum. All objects emit and absorb radiative heat energy based on their temperature, with hotter objects emitting more radiation. Radiation is a key consideration in high-temperature applications, such as furnaces, solar energy systems, and thermal insulation for spacecraft.

Understanding these mechanisms is crucial for accurately modeling and predicting heat transfer in various applications, from industrial processes to everyday scenarios like heating a room or cooking food. By considering the interplay between conduction, convection, and radiation, engineers and scientists can develop more efficient and effective thermal management solutions.

2.1.1.3 Practical Applications and Importance

Heat conduction is pivotal in numerous practical applications across various fields:

Engineering: In mechanical and civil engineering, understanding heat conduction is essential for designing and analyzing systems like heat exchangers, thermal insulation for buildings, and cooling systems for electronic devices. Efficient thermal management is critical for optimizing performance, ensuring safety, and extending the lifespan of engineered systems.

Environmental Science: Heat conduction plays a role in natural processes such as geothermal energy transfer, soil temperature regulation, and the thermal behavior of natural water bodies. Understanding these processes is vital for assessing environmental impacts, predicting climate changes, and managing natural resources.

Everyday Life: Common activities like cooking, using thermal insulation in clothing, and heating or cooling living spaces all rely on principles of heat conduction. By understanding these principles, individuals can make informed decisions about energy usage and efficiency, contributing to sustainability and comfort in daily life.[4]

By studying heat conduction, we can optimize the efficiency and effectiveness of thermal management in various systems, leading to advancements in technology and improvements in quality of life. Understanding the principles of heat conduction also supports innovation in emerging fields such as nanotechnology, renewable energy, and advanced materials science.In summary, heat conduction is a vital process that underpins many natural and engineered systems. The principles governing heat conduction, such as Fourier's law, along with material properties like thermal conductivity, are fundamental to understanding and optimizing heat transfer in diverse applications. By integrating these principles into Physics-Informed Neural Networks, researchers can leverage machine learning techniques to solve complex heat conduction problems with greater accuracy and efficiency.

2.2 Introduction to Neural Networks

Neural networks (NNs) are a foundational technology in the field of artificial intelligence, consisting of interconnected units or neurons that work together to perform complex computations. These networks have revolutionized how we approach problem-solving in various domains, from image recognition to natural language processing. By mimicking the structure and function of the human brain, neural networks have the ability to learn patterns and representations from vast amounts of data, making them indispensable tools in the realm of machine learning and deep learning.

Neural networks are composed of layers of neurons, each performing mathematical transformations of their inputs to solve complex problems. These networks are extensively used in deep learning and machine learning, which are subfields of artificial intelligence. A typical artificial neural network (ANN) consists of an input layer, one or more hidden layers, and an output layer, as illustrated in Figure 2.2. The network is composed of nodes arranged in parallel layers, where each node is connected to nodes in the adjacent layer, with each connection having an associated weight $w_{ij}^{(l)} \in \mathbb{R}$.[5]

The architecture of a neural network is crucial for its performance, with each layer serving a specific purpose in the transformation of input data into meaningful output. The input layer receives the raw data, which is then processed by the hidden layers. These hidden layers are where the network learns to recognize patterns and features through the use of weights and biases. Finally, the output layer provides the final prediction or classification based on the learned representations.

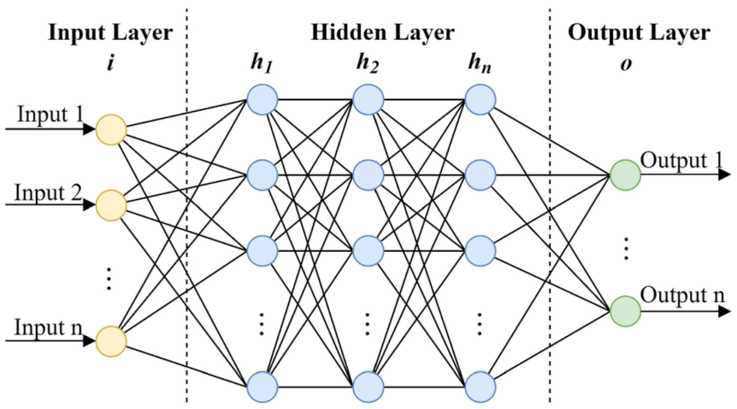


Figure 2.2: Neural Network architecture

Each node in the network computes a weighted sum of its inputs and produces an output based on its associated activation function σ, as depicted in equation (2). Here, N_l represents the number of nodes in layer l, $\{x_i\}_{i=1}^{N_{l-1}}$ is the set of inputs from the nodes in the previous layer $l-1$, and b_l is the bias associated with layer l. The bias is introduced at each node to adjust the activation function, thereby helping to produce the desired output. The final prediction of the network is denoted by $\hat{y}$. This process is commonly referred to as forward propagation and is given by

$$\hat{y} = \sigma\left(\sum_{j=1}^{N_l}\left(\sum_{i=1}^{N_{l-1}} w_{ij}^{(l)} x_i + b_j^{(l)}\right)\right)$$

The activation function introduces non-linearity into the network, enabling it to approximate complex nonlinear functions. These functions are crucial in determining the accuracy of a model and the computational efficiency of training a model. They have a significant effect on the network's ability to converge, i.e., to find the optimal weights w and biases b. Without this non-linearity, a multilayered network would essentially perform as a single-layered network, since it would only be capable of linear combinations of the input functions.[5]

Common activation functions include the sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). Each function has its own advantages and is suitable for different types of tasks. For instance, the ReLU function is widely used in deep networks due to its ability to mitigate the vanishing gradient problem, which can occur with sigmoid and tanh functions in deep architectures.[6]

2.2.1 Types of Neural Networks

Beyond standard feedforward networks, other important types include:

1. **Convolutional Neural Networks (CNNs):** Designed for processing grid-like data (e.g., images). The convolution operation is defined as

$$(f * g)(x, y) = \sum_m \sum_n f(m, n) g(x - m, y - n)$$

where f is the input (e.g., an image) and g is the kernel (or filter). CNNs utilize layers of convolutions followed by pooling layers to reduce dimensionality while preserving spatial hierarchies. These networks are particularly effective for tasks involving visual data, such as image classification, object detection, and image segmentation.

2. **Recurrent Neural Networks (RNNs):** It is designed for processing sequential data using hidden states and is give by

$$h_t = \sigma(W_{xh} x_t + W_{hh} h_{t-1} + b_h)$$

where h_t is the hidden state at time t, and x_t is the input at time t. RNNs maintain a hidden state that captures information from previous time steps, making them suitable for tasks involving sequences, such as language modeling and time series prediction. Their ability to handle temporal dependencies makes them ideal for applications like speech recognition, machine translation, and video analysis.

3. **Generative Adversarial Networks (GANs):** Comprising a generator and a discriminator, GANs are designed to generate new, synthetic instances of data that can pass for real data. The generator creates fake data, while the discriminator evaluates its authenticity and is given by

$$\text{min}_G \text{max}_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}}[\text{log}\, D(x)] + \mathbb{E}_{z \sim p_z}[\text{log}(1 - D(G(z)))]$$

where $G(z)$ produces samples from a random noise distribution z, and D is the discriminator that classifies samples as real or fake. GANs have gained popularity for their ability to generate high-quality synthetic data, making them useful for applications such as image generation, style transfer, and data augmentation.[6]

These specialized architectures are tailored for specific data types and problems, offering improved performance over standard feedforward networks in their respective domains. Each type of architecture leverages unique structural properties to optimize learning for particular tasks, showcasing the versatility of neural networks in solving diverse problems across various fields. By understanding the strengths and limitations of each architecture, practitioners can select and design neural networks that best suit their specific needs and challenges.

2.2.2 Training Process

The training of neural networks is primarily accomplished through the backpropagation algorithm, which efficiently computes the gradient of the loss function with respect to the network parameters. This process involves several key steps which are mentioned below.

1. **Forward Pass**
In this step, we compute the output for each layer $l = 1, \dots, L$ of the neural network. The forward pass involves propagating the input through the layers, where each layer applies a transformation using weighted connections and biases, followed by a non-linear activation function. This process allows the network to learn complex representations of the input data. The transformation is expressed as

$$a^{(l)} = \sigma(z^{(l)}) = \sigma(W^{(l)}a^{(l-1)} + b^{(l)})$$

In this equation, $a^{(l)}$ represents the activations of the l-th layer, $W^{(l)}$ denotes the weight matrix, $b^{(l)}$ is the bias vector, and σ is the activation function. The transformation captures the interactions between features and enables the network to extract meaningful patterns from the input.

2. **Compute the Output Error**
After obtaining the output from the final layer, we need to assess how far off the predictions are from the actual target values. This is done by calculating the output error, which quantifies the difference between the predicted and actual outputs. The output error is calculated as

$$\delta^{(L)} = \nabla_a C \odot \sigma'(z^{(L)})$$

Here, $\nabla_a C$ represents the gradient of the cost function with respect to the output activations. The term $\sigma'(z^{(L)})$ is the derivative of the activation function at the output layer, indicating how sensitive the output is to changes in the input. The element-wise multiplication $\odot$ allows us to compute the error at the output layer, which is essential for assessing the network's performance and guiding the subsequent updates.

3. **Backpropagate the Error**
To optimize the network, we must propagate the output error back through the layers. This step involves calculating the error for each layer starting from the output layer and moving backward. The error for each layer can be expressed as

$$\delta^{(l)} = \left((W^{(l+1)})^T \delta^{(l+1)}\right) \odot \sigma'(z^{(l)})$$

Here, $\delta^{(l)}$ represents the error for the l-th layer. The error from the subsequent layer is weighted by the transpose of the weight matrix, and the derivative of the activation function is applied to maintain the correct direction of the gradient. This process is crucial for determining how each neuron contributed to the final error, allowing for targeted adjustments in the network's parameters.

4. **Compute the Gradients**
After calculating the errors for each layer, we need to compute the gradients of the cost function with respect to the weights and biases. The gradients are given by

$$\frac{\partial C}{\partial W^{(l)}} = \delta^{(l)}(a^{(l-1)})^T, \quad \frac{\partial C}{\partial b^{(l)}} = \delta^{(l)}$$

In this step, the gradients of the cost function with respect to weights and biases are computed. These gradients will be used to update the weights and biases in the next step. The computation of gradients is a central aspect of the training process, as it directly influences the network's ability to learn from data.

Optimization techniques like gradient descent and its variants are used to update the network parameters. The standard gradient descent update rule is

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t)$$

where η is the learning rate, controlling how much to adjust the weights with respect to the gradient, and $J(\theta)$ is the cost function representing the error of the model. The choice of learning rate is critical, as it affects the speed and stability of the training process. A learning rate that is too high can cause the model to diverge, while a learning rate that is too low can result in slow convergence.

Variants of gradient descent include :

1. **Stochastic Gradient Descent (SGD)**
In this approach, the weights are updated using the gradient calculated from a single training example. The update rule for this method is

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J_i(\theta_t)$$

In SGD, the weights are updated using the gradient calculated from a single training example J_i, allowing for faster updates but with more noise. This approach is advantageous in scenarios with large datasets, as it allows the model to update more frequently and potentially escape local minima.

2. **Adam (Adaptive Moment Estimation)**
Adam is an advanced optimization algorithm that combines the benefits of two extensions of stochastic gradient descent. The equations for Adam are

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta J(\theta_t)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\nabla_\theta J(\theta_t)\right)^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t$$

Adam utilizes moving averages of both the gradients m_t and the squared gradients v_t to adapt the learning rate for each parameter, effectively incorporating momentum and scaling based on past gradients. This method is particularly effective for handling sparse gradients and non-stationary objectives, making it a popular choice for training deep networks.[6]

2.2.3 Regularization Techniques

Regularization techniques are crucial for preventing overfitting in neural networks, which occurs when the model learns noise in the training data rather than the underlying distribution. Overfitting can lead to poor generalization performance, where the model performs well on training data but poorly on unseen data. Key methods include following.

1. L1/L2 Regularization
This technique involves adding a penalty term to the loss function to discourage overly complex models. The penalty varies based on the type of regularization used, either promoting sparsity or smaller weights.It is given by

$$J(\theta) = J_0(\theta) + \lambda R(\theta)$$

where $R(\theta)$ is the regularization term. The regularization parameter λ controls the strength of the penalty. - L1: $R(\theta) = \|\theta\|_1 = \sum_i |\theta_i|$ promotes sparsity in the parameter space, effectively reducing the complexity of the model by encouraging many parameters to be zero. - L2: $R(\theta) = \|\theta\|_2^2 = \sum_i \theta_i^2$ promotes smaller weights and smoothens the model, which can help in reducing the model's sensitivity to small fluctuations in the data.

2. Dropout
This regularization technique involves randomly setting a fraction p of input units to 0 during training. By doing this, dropout prevents neurons from co-adapting too much, leading to more robust feature learning. It is expressed as

$$y = f(Wx) \to y = f(W(x \odot r))$$

where $r \sim \text{Bernoulli}(p)$. This technique prevents co-adaptation of neurons, thereby making the network more robust. By randomly dropping units during training, dropout introduces noise that forces the network to learn more distributed representations, improving its ability to generalize.

3. Early Stopping
This method involves monitoring the validation error during training and stopping the process when the validation error begins to increase. This approach helps prevent overfitting by ensuring that the model does not train for too long. It is expressed as

$$E_{\text{val}}(t+1) > E_{\text{val}}(t) \text{ for } t > t^*$$

where t^* is the optimal stopping point. This helps to ensure that training is halted before the model begins to overfit to the training data. By using a separate validation set to monitor performance, early stopping provides a simple yet effective way to determine the best model iteration.

These techniques help prevent overfitting by reducing model complexity, introducing noise, or limiting training time, respectively. By incorporating regularization methods, practitioners can achieve models that generalize well to new, unseen data, enhancing the reliability and robustness of neural network applications.[7]

2.3 Introduction to Physics-Informed Neural Networks (PINNs)

Physics-Informed Neural Networks (PINNs) are an advanced type of neural network that incorporate known physical laws into the learning process, enabling the model to adhere to these laws when making predictions. This integration is particularly useful in scenarios where traditional data-driven models may fail due to data scarcity or the complex nature of physical systems.

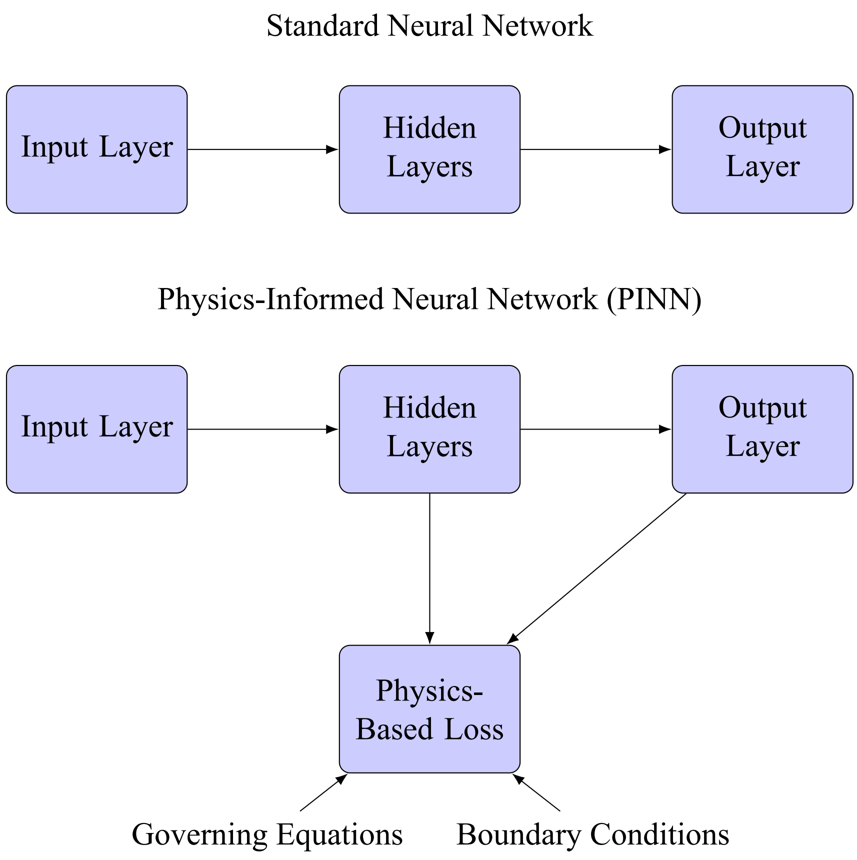


Figure 2.3: Standard Neural Network and Physics-Informed Neural Network (PINN)

The foundational principle of PINNs is to embed physical laws, typically expressed as differential equations, directly into the architecture of neural networks. This is achieved by constructing a custom loss function that not only penalizes the prediction error but also the deviation from the physical laws that is

$$L = L_{data} + \lambda L_{physics}$$

where L_{data} is the loss due to prediction error, $L_{physics}$ is the loss due to deviation from physical laws, and λ is a regularization parameter that controls the trade-off between these two losses [8].

Figure 2.4 illustrates the architecture of a Physics-Informed Neural Network. The network takes input coordinates (x) and outputs predictions (u). The key feature of PINNs is the incorporation of physical laws into the loss function. This is achieved by computing gradients of the network's output with respect to its input $(\partial u/\partial x, \partial^2 u/\partial x^2$, etc.$)$ and using these to evaluate the residual of the underlying differential equation. This residual is then added as an extra term in the loss function, ensuring that the learned solution is consistent with the known physics [9].

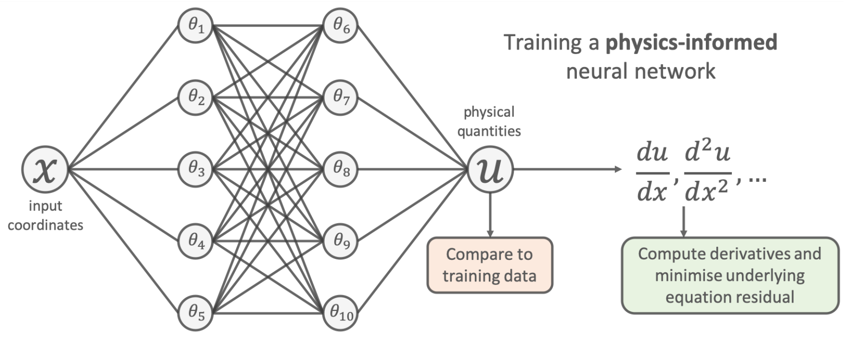


Figure 2.4: Schematic representation of a Physics-Informed Neural Network

PINNs have been applied across various domains, including fluid dynamics, quantum mechanics, and material science, to solve problems that are otherwise computationally expensive or infeasible to solve with traditional numerical methods. The ability of PINNs to use both data and laws of physics makes them uniquely capable of solving inverse problems, where parameters within the laws need to be inferred from observational data. One of the key advantages of PINNs is their ability to handle complex, nonlinear partial differential equations (PDEs) that are often challenging for traditional numerical methods. By encoding the PDE into the loss function, PINNs can find solutions that satisfy both the data and the governing equations simultaneously.

To demonstrate the effectiveness of PINNs, consider a simple example of a damped harmonic oscillator. The underlying physics can be described by the following differential equation

$$m\frac{d^2u}{dt^2} + \mu\frac{du}{dt} + ku = 0$$

where m is the mass of the oscillator, μ is the coefficient of friction, and k is the spring constant. For a PINN to solve this problem, we would construct the following loss function

$$L = \frac{1}{N}\sum_{i=1}^{N} |u(t_i) - u_i|^2 + \lambda\frac{1}{N_f}\sum_{j=1}^{N_f} |m\frac{d^2u}{dt^2}(t_j) + \mu\frac{du}{dt}(t_j) + ku(t_j)|^2$$

where the first term represents the mean squared error between the network's predictions and the available data points, and the second term enforces the physics of the damped harmonic oscillator [9].

Physics-Informed Neural Networks (PINNs) offer significant advantages in terms of computational efficiency. Unlike traditional numerical methods that often require fine spatial and temporal discretization, PINNs can provide continuous solutions in both space and time. This continuous representation allows for efficient evaluation of the solution at any point in the domain without the need for interpolation. Moreover, PINNs have shown remarkable capabilities in handling inverse problems and parameter estimation. In many scientific and engineering applications, the parameters of the governing equations are unknown and need to be inferred from data. PINNs can naturally handle such scenarios by treating these unknown parameters as trainable variables within the network.

Traditional numerical methods, including the Finite Difference Method (FDM) and Finite Element Method (FEM), are widely employed for solving partial differential equations (PDEs) that arise in numerous fields such as fluid dynamics, structural analysis, and heat conduction. These methods operate by discretizing the governing equations and the computational domain into a mesh or grid.

FDM approximates derivatives using difference equations, which can lead to straightforward implementations for simple geometries. On the other hand, FEM subdivides the domain into smaller, manageable elements, allowing for greater flexibility in handling complex geometries and varying material properties. While these traditional methods are well-established and provide reliable solutions, they often require fine spatial and temporal discretization to achieve high accuracy, which can lead to significant computational costs. Additionally, they necessitate specialized techniques for boundary and initial conditions and may struggle with problems involving sharp gradients or discontinuities. Despite these limitations, traditional methods remain a cornerstone in numerical analysis due to their rigorous mathematical foundations and extensive error quantification techniques.

The following table summarizes the advantages and limitations of traditional methods (FDM and FEM) and PINNs for heat conduction problems.

Criteria	**Traditional Methods (FDM, FEM)**	**PINNs**
Mesh Requirement	Requires explicit mesh generation	Mesh-free approach
Physics Integration	Implicit in discretization	Explicit through loss function
Boundary Conditions	Handled separately for each case	Unified handling within loss function
Flexibility	Less adaptable to complex geometries	Highly flexible for various conditions
Error Quantification	Established techniques available	Still under research
Computational Efficiency	Can be optimized for speed	May be computationally intensive
Handling of Discontinuities	May struggle with sharp changes	Can struggle without adaptive techniques

Table 2.1: Comparative Analysis of Traditional Methods and PINNs

2.4 2D Heat Conduction Equation

The two-dimensional heat conduction equation is a pivotal partial differential equation (PDE) in the field of thermal physics and engineering. This equation serves as a mathematical model to describe the temporal and spatial distribution of temperature in a two-dimensional medium. It is crucial for understanding how heat propagates within materials and across various systems, impacting a wide range of applications from industrial processes to environmental science.

The origin of the 2D heat conduction equation lies in the principles of energy conservation and Fourier's law of heat conduction. Energy conservation dictates that the rate of change of internal energy within a body must equal the net rate of heat entering the body. Fourier's law, on the other hand, establishes that the heat flux through a material is proportional to the negative gradient of temperature, meaning that heat naturally flows from hotter to cooler regions. This foundational understanding is pivotal in formulating the equation that governs heat conduction in two dimensions.[10]

For a homogeneous and isotropic medium—where the material properties are uniform and identical in all directions—and assuming constant thermal properties, the 2D heat conduction equation is expressed as

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{2}$$

where

- $u(x, y, t)$ represents the temperature at a specific position (x, y) and time t. This function captures the thermal state of the medium and its evolution over time, providing a comprehensive view of how temperature changes in response to internal and external factors.
- α denotes the thermal diffusivity of the medium, a crucial parameter that quantifies how rapidly heat diffuses through the material. It is a measure of the material's ability to conduct heat relative to its ability to store heat.
- $\frac{\partial u}{\partial t}$ signifies the rate of change of temperature with respect to time, indicating the dynamic evolution of the thermal state. This term is essential for understanding transient heat conduction, where temperature changes occur over time.
- $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ are the second-order partial derivatives of temperature with respect to the spatial coordinates x and y, respectively. These derivatives capture the curvature of the temperature distribution, reflecting how temperature gradients influence heat flow within the medium.[11]

The thermal diffusivity α is defined as

$$\alpha = \frac{k}{\rho c_p}$$

where k is the thermal conductivity, ρ is the density, and c_p is the specific heat capacity of the material. This relation underscores that thermal diffusivity is influenced by the material's ability to conduct heat, its density, and its capacity to store thermal energy. A higher thermal diffusivity indicates that the material can quickly adjust its temperature in response to changes, leading to faster heat distribution throughout the medium.

Equation 2 encapsulates the fundamental principle that the rate of change of temperature at any point within the medium is proportional to the spatial curvature of the temperature field at that point. This relationship enables the prediction of heat flow and distribution within a two-dimensional medium over time. More specifically, it implies that regions of high temperature will transfer heat to adjacent cooler regions, thereby promoting a more uniform temperature distribution as time progresses. This smoothing effect, characteristic of diffusion processes, is central to the analysis of thermal systems and plays a crucial role in numerous applications.

In situations where the temperature distribution reaches a steady state and does not change with time ($\frac{\partial u}{\partial t} = 0$), the heat conduction equation simplifies to the Laplace equation which is given by

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

The Laplace equation is particularly useful in analyzing long-term thermal behavior in systems where transient effects have dissipated. In the steady state, the heat conduction process stabilizes, and the temperature distribution can be analyzed without the complicating factor of time dependence. This simplification is invaluable in scenarios where systems have reached equilibrium, such as in many engineering applications where constant boundary conditions are maintained over extended periods.

The 2D heat conduction equation finds applications across a wide spectrum of fields, each presenting unique challenges and opportunities.

1. **Thermal Management in Electronic Devices:** Effective thermal management is critical in the design and operation of electronic devices, such as microprocessors, batteries, and power electronics, where excessive heat can lead to reduced performance, reliability issues, or even catastrophic failure. Understanding heat conduction allows engineers to design cooling systems, such as heat sinks and thermal interfaces, that maintain components within safe temperature limits.[10]

2. **Building Insulation and Energy Efficiency Studies:** In the context of sustainable architecture and energy conservation, analyzing heat conduction through building materials is essential for optimizing thermal performance. By understanding how heat is lost or gained through walls, roofs, and windows, architects and engineers can design structures that minimize energy consumption while maximizing comfort for occupants.

3. **Geothermal Energy Systems:** In geothermal energy applications, the 2D heat equation helps model the transfer of thermal energy from the Earth's subsurface to the surface, informing the design and optimization of geothermal power plants. Accurate modeling of heat conduction in geological formations enhances the efficiency of energy extraction techniques and supports the development of sustainable energy sources.

4. **Climate Modeling and Weather Prediction:** Simulating temperature distributions in the atmosphere and oceans is vital for understanding climate dynamics and forecasting weather patterns. The 2D heat equation plays a crucial role in climate models that predict temperature changes and their impacts on global and regional scales, providing valuable insights for environmental policy and planning.

5. **Materials Science and Manufacturing Processes:** In materials science, heat treatment processes such as annealing, quenching, and welding are used to alter material properties. Understanding heat conduction is essential for controlling these processes and achieving

desired outcomes, such as improved mechanical properties, reduced internal stresses, or enhanced microstructural characteristics.[11]

Various methods can be employed to solve the 2D heat equation, each offering distinct advantages and limitations:

1. **Analytical Methods:** Techniques such as separation of variables provide exact solutions for simple geometries and boundary conditions. These methods often involve transforming the PDE into a set of ordinary differential equations that are solvable under given conditions. While analytical solutions offer precise results, they are typically limited to cases with simple geometries, homogeneous material properties, and straightforward boundary conditions.

2. **Numerical Methods:** Numerical approaches, including finite difference, finite element, and finite volume methods, discretize the spatial domain and approximate the solution at grid points. These methods are versatile and well-suited for complex geometries, heterogeneous materials, and intricate boundary conditions. Numerical methods are widely used in practice due to their flexibility and ability to handle real-world complexities, although they often require significant computational resources and careful attention to numerical stability and convergence.

3. **Transform Methods:** Techniques such as Fourier and Laplace transforms are effective for certain types of problems with specific boundary or initial conditions. These methods convert the PDE into algebraic equations, which can be solved more easily and then transformed back to obtain the solution in the original domain. Transform methods are particularly useful for problems with periodic or semi-infinite domains, allowing for efficient analysis of systems with repetitive or large-scale features.[11]

In this thesis, we focus on solving the 2D heat equation using Physics-Informed Neural Networks (PINNs), particularly with Neumann boundary conditions. PINNs represent a novel approach that combines the flexibility of neural networks with the physical constraints imposed by the heat equation, offering a powerful tool for solving complex heat conduction problems. The use of PINNs enables the incorporation of physical laws directly into the training process, allowing the model to learn from both data and the governing equations simultaneously. This integration of data-driven and physics-based methodologies provides a robust framework for addressing the challenges of heat conduction in heterogeneous and complex media.Understanding and solving the 2D heat equation is crucial for accurately predicting temperature distributions and heat flow in complex systems. This knowledge enables better design and optimization of thermal processes and technologies, leading to more efficient and effective solutions in various engineering and scientific domains. By leveraging advanced computational techniques and innovative modeling approaches like PINNs, we aim to enhance our ability to model and control thermal phenomena, ultimately advancing our capacity to develop cutting-edge technologies and sustainable energy solutions. Through this comprehensive exploration of the two-dimensional heat conduction equation, we gain valuable insights into the fundamental principles governing heat transfer, the mathematical tools available for analysis, and the wide array of practical applications that benefit from a deeper understanding of thermal dynamics. This expanded perspective empowers researchers and practitioners to tackle complex thermal challenges with confidence and creativity, paving the way for innovations that enhance our quality of life and address pressing global issues.

2.5 Boundary Conditions in Heat Conduction Problems

Boundary conditions are fundamental components in the analysis of heat conduction problems, as they define the thermal behavior at the boundaries of the domain under consideration. These conditions, together with the governing differential equation and initial conditions, form a complete mathematical framework for the heat transfer problem. They are crucial for accurately representing how a system exchanges heat with its surroundings and for predicting the temperature distribution within the domain.[12]

In the context of the two-dimensional heat conduction equation, multiple types of boundary conditions can be applied, each corresponding to different physical scenarios and constraints. These conditions are essential for ensuring that the solution to the heat conduction equation aligns with the physical reality of the problem being modeled. By understanding and correctly implementing these boundary conditions, one can achieve a realistic and reliable simulation of heat transfer processes.

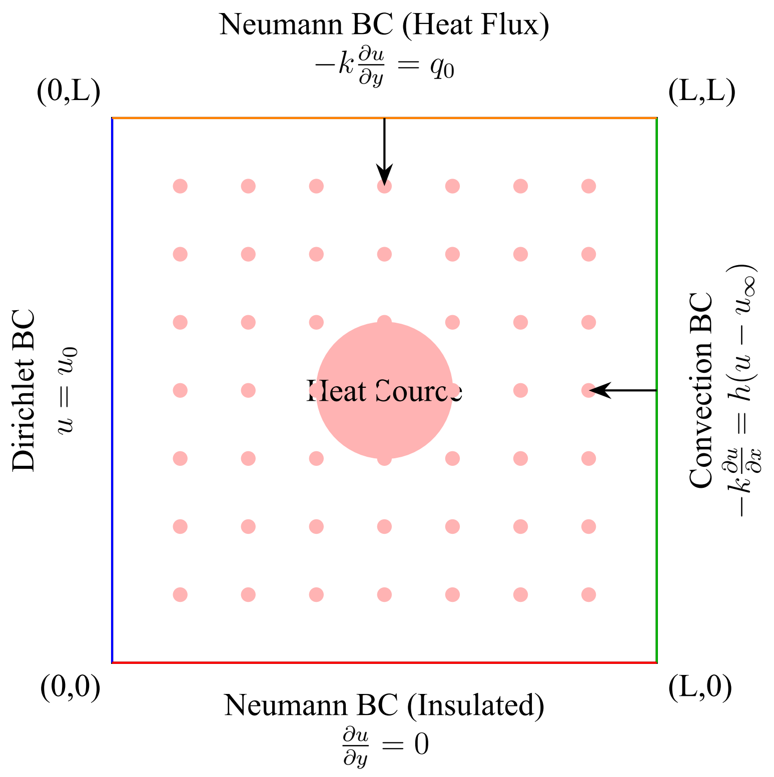

Neumann BC (Heat Flux)
$-k\frac{\partial u}{\partial y} = q_0$
(0,L)
(L,L)
Dirichlet BC
$u = u_0$
Heat Source
Convection BC
$-k\frac{\partial u}{\partial x} = h(u - u_\infty)$
(0,0)
(L,0)
Neumann BC (Insulated)
$\frac{\partial u}{\partial y} = 0$

Figure 2.5: Boundary Conditions in Heat Conduction

Figure 2.5 presents a schematic representation of various boundary conditions, including the Neumann condition, applicable to a heat conduction problem. To study the these boundary conditions, particularly Neumann conditions, is vital for effectively modeling heat transfer processes and developing solutions using advanced computational techniques like Physics-Informed Neural Networks (PINNs).

2.5.1 Dirichlet Boundary Condition

The Dirichlet boundary condition, also known as the first-type boundary condition, specifies the temperature at the boundary of the domain. It is mathematically expressed as

$$u(x, y, t) = f(x, y, t)$$

where $f(x, y, t)$ is a known function that defines the temperature along the boundary over time. This condition is particularly useful when the surface temperature is controlled or known, such as in situations where the boundary is in direct contact with a thermal reservoir or a regulated heat source.

In practical applications, Dirichlet boundary conditions are extensively used in engineering and environmental systems where precise temperature control is necessary. For example, in the design and operation of HVAC (Heating, Ventilation, and Air Conditioning) systems, maintaining a constant temperature on interior walls is often required to ensure thermal comfort and energy efficiency. Similarly, in electronic cooling systems, components such as heat sinks are designed to maintain specific temperatures to ensure the reliability and performance of electronic devices.

The Dirichlet condition is straightforward to implement in both analytical and numerical solutions, as it directly specifies the temperature values at the boundaries. This simplicity makes it a preferred choice in many scenarios where the boundary temperature is a known quantity.

2.5.2 Robin Boundary Condition

The Robin boundary condition, also known as the mixed or third-type boundary condition, combines elements of both Dirichlet and Neumann conditions. It is typically used to model convective heat transfer at a boundary and is mathematically expressed as

$$-k\frac{\partial u}{\partial n} = h(u - u_\infty)$$

In this expression, k is the thermal conductivity of the material, h is the convective heat transfer coefficient, and u_∞ is the ambient temperature of the surrounding fluid. The Robin condition is applicable in scenarios where heat transfer at the boundary involves both conduction and convection, such as a solid surface exposed to a moving fluid.[13]

This condition is essential in many industrial and environmental applications where convective heat transfer plays a significant role. For instance, in the design of heat exchangers, the Robin condition can model the thermal exchange between the fluid and the solid surfaces, providing insights into the efficiency and effectiveness of the heat transfer process. Similarly, in climate modeling, the Robin condition can represent heat transfer between the Earth's surface and the atmosphere, allowing for the simulation of temperature changes due to natural and anthropogenic factors.

Implementing Robin boundary conditions in numerical models requires careful consideration of both conductive and convective heat transfer mechanisms, making it a versatile yet complex condition to apply.

2.5.3 Periodic Boundary Condition

Periodic boundary conditions are used when the domain exhibits cyclic or repeating behavior, such as in the analysis of heat transfer in a rotating or repetitive system. These conditions ensure that the temperature and its gradient are continuous across the periodic boundary, effectively modeling systems where the geometry or thermal behavior repeats itself.

In practical applications, periodic boundary conditions are applied in simulations of materials with repeating structures, such as crystals, composites, or textiles. They are also used in computational fluid dynamics (CFD) to model flow patterns in rotating machinery, such as turbines and compressors, where the thermal and flow behaviors are periodic.

Applying periodic boundary conditions can significantly reduce the computational complexity of a problem by allowing the analysis of a representative section of the domain rather than the entire system. This approach is particularly advantageous in large-scale simulations where computational resources are limited.

2.5.4 Neumann Boundary Condition

The Neumann boundary condition, or second-type boundary condition, specifies the heat flux at the boundary of the domain. It is mathematically represented as

$$-k\frac{\partial u}{\partial n} = q(x, y, t)$$

In this equation, k is the thermal conductivity, $\frac{\partial u}{\partial n}$ is the temperature gradient normal to the boundary, and $q(x, y, t)$ is the specified heat flux. The Neumann condition is particularly relevant in scenarios where the heat flow at the boundary is known or controlled, such as insulated surfaces or areas with constant heat input.

In this thesis, we primarily focus on Neumann boundary conditions due to their wide applicability in various engineering and scientific problems. The Neumann condition is especially useful in cases where heat flux, rather than temperature, is the known or controlled parameter at the boundaries.[12]

Key aspects of Neumann boundary conditions include

1. Physical Interpretation: The Neumann condition represents a specified heat flux at the boundary, which can model various real-world scenarios such as insulated surfaces (zero heat flux), constant heat input, or radiative heat transfer. For instance, in electronic devices, understanding the heat dissipation rate on the surface is crucial for effective thermal management.

2. Mathematical Formulation: The condition is expressed in terms of the normal derivative of temperature at the boundary, aligning with Fourier's law of heat conduction. This law relates heat flux to the temperature gradient, providing a direct link between the physical process of heat transfer and its mathematical representation.

4. Applications: Neumann conditions are widely used in the thermal management of electronic devices, building energy analysis, geothermal studies, and materials processing. For example, in the analysis of heat dissipation in electronic components, the heat flux at the surface of a chip can be specified as a Neumann condition. Similarly, in geothermal energy systems, the heat flux at the Earth's surface can be modeled to understand subsurface temperature dynamics.

5. Importance in Physics-Informed Neural Networks: When implementing PINNs for heat

conduction problems, incorporating Neumann boundary conditions requires careful consideration in the loss function formulation. The network must learn to satisfy these flux conditions at the boundaries while also solving the heat equation in the interior of the domain. This involves designing loss functions that appropriately balance the physics-based constraints with data-driven insights, ensuring that the model accurately represents the underlying physical processes.

2.6 PyTorch Framework for PINNs

The implementation of Physics-Informed Neural Networks (PINNs) can be significantly enhanced using the PyTorch framework, which offers a flexible and efficient platform for developing deep learning models. As an open-source machine learning library, PyTorch is distinguished by its dynamic computational graph, allowing for seamless integration of neural network components and automatic differentiation capabilities. This feature is particularly advantageous for PINNs, where the computation of derivatives is essential for incorporating governing physical laws directly into the network.

One standout feature of PyTorch is its dynamic graph structure, which enables easy modification of the network architecture during runtime. This capability is crucial for rapid experimentation and prototyping of PINN models, allowing researchers to tailor the networks to specific problem domains effectively. Researchers can adjust various parameters, such as the number of layers to capture the complexity of the problem, the number of neurons to optimize performance and improve accuracy, and the choice of activation functions to introduce necessary non-linearity for specific applications. Furthermore, PyTorch's extensive library of pre-built modules and functions simplifies the implementation of complex neural network architectures, making the framework accessible to both researchers and practitioners.

The integration of PINNs with PyTorch is further bolstered by specialized libraries like PINNs-Torch, which provide essential tools and utilities that streamline the development process. These libraries facilitate efficient data handling, allowing for the organization and preprocessing of datasets. Additionally, they support customizable loss functions, enabling users to encode physical laws, initial conditions, and boundary constraints directly into the training process. They also offer support for various types of boundary and initial conditions, making it easier to model complex physical scenarios. Such capabilities allow researchers to focus on core aspects of their investigations, minimizing the complexities typically associated with neural network implementation.

Another notable advantage of the PyTorch framework is its built-in support for GPU acceleration, which ensures that PINN models can be trained efficiently on large datasets or complex simulations. This feature significantly reduces computation time, enhancing scalability and allowing researchers to tackle high-dimensional problems or those requiring extensive computational resources. Applications of this capability span across fields such as fluid dynamics, where researchers simulate the behavior of fluid flows under various conditions, and structural analysis, where they assess the integrity and performance of materials under stress. The ability to leverage GPU resources is essential for effectively solving computationally intensive problems in various scientific and engineering fields.

Physics-Informed Neural Networks have demonstrated remarkable versatility in addressing a wide range of physical problems, such as heat conduction in multiple dimensions. They are particularly effective in managing:

1. Steady-state problems, where systems reach a stable temperature distribution over time.
2. Transient problems, where temperature varies with both time and space.
3. Complex boundary conditions, including Neumann conditions that specify heat flux at the boundaries.
4. Source terms or forcing functions that model external influences on the system.

By enabling the flexible formulation of custom loss functions, PINNs effectively encode these physical laws, initial conditions, and boundary constraints. Additionally, when combined with modern visualization techniques, PINNs can provide insightful representations of how solutions evolve over time and space. This visualization capability is crucial for understanding the dynamics of the modeled phenomena and effectively conveying results to stakeholders.

In summary, the PyTorch framework serves as a robust and versatile platform for implementing PINNs. Its dynamic graph structure, comprehensive library, and GPU support contribute significantly to enhancing both the speed and usability of these models. Consequently, PyTorch has become an ideal choice for researchers aiming to develop and deploy sophisticated PINN-based solutions across various scientific and engineering domains [14].

3 Case Studies

3.1 2D Heat Conduction with Center Heat Source

This case study explores the application of Physics-Informed Neural Networks (PINNs) to address a challenging two-dimensional heat conduction problem characterized by a localized heat source. The investigation is set within a well-defined square domain, chosen for its symmetry and simplicity, which allows for a focused study of heat dynamics. Within this domain, the boundaries are assumed to be perfectly insulated, meaning that no heat can escape or enter from the edges. This setup is akin to a closed thermal system where energy transformation occurs internally without external influence. A centralized heat source is introduced at the heart of the domain, simulating conditions where heat is generated from a point or region of interest, such as in electronic circuit hotspots or localized heating applications.

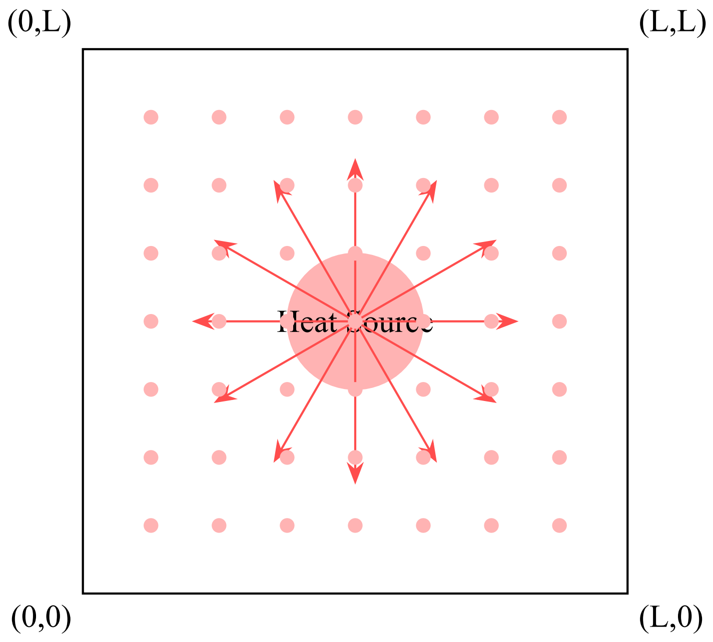


Figure 3.1: 2D Heat Conduction with Center Heat Source

Through this study, we aim to demonstrate the capability of PINNs to solve complex, time-dependent partial differential equations (PDEs) that incorporate specific boundary conditions and source terms. PINNs leverage the strengths of neural networks to embed the governing physical laws into the learning process, efficiently approximating solutions even in challenging scenarios. Understanding the dynamics of heat diffusion from a localized source is crucial across various domains.

3.1.1 Governing Equation

The heat conduction process in this scenario is governed by the following partial differential equation, which mathematically describes how the temperature field evolves over time and space within the domain and is expressed as

$$\frac{\partial u}{\partial t} = \epsilon \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + f(x, y, t)$$

where, the function $u(x, y, t)$ denotes the temperature distribution across the spatial coordinates x and y at any given time t. The parameter ϵ, representing thermal diffusivity, is set to 0.1 in this problem. Thermal diffusivity is a key material property that characterizes the rate at which heat spreads through a material. A higher diffusivity indicates that heat is conducted quickly, while a lower value suggests slower conduction. In our study, setting ϵ to 0.1 reflects a moderate rate of heat diffusion, suitable for capturing the nuanced interplay between conduction and localized heating. The term $f(x, y, t)$ represents the heat source, which introduces energy into the system. This source term adds complexity to the model by providing localized energy input that influences the temperature distribution. The PDE effectively combines the classical parabolic heat equation, which describes the natural diffusion of heat throughout a medium, with a source term that accounts for the direct thermal input from the localized heat source. This formulation is essential for accurately modeling scenarios where heat is not uniformly distributed but rather concentrated at specific points or regions, reflecting real-world conditions encountered in various applications, from industrial manufacturing processes to natural systems.

3.1.2 Domain and Boundary Conditions

The problem is defined within a square domain $\Omega = [0, L] \times [0, L]$, where $L = 1.0$ denotes the length of each side, establishing a confined space for the heat conduction analysis. This choice of a square domain simplifies the computational setup while allowing for a comprehensive study of heat distribution and boundary interactions. To accurately model this physical scenario, Neumann boundary conditions are applied to all sides of the domain, representing insulated boundaries as

$$\frac{\partial u}{\partial n} = 0 \quad \text{on } \partial\Omega$$

Here, n represents the outward normal vector to the boundary $\partial\Omega$. These Neumann boundary conditions imply that there is no net heat flux across the boundaries of the domain, effectively simulating a perfectly insulated system where heat cannot escape or be absorbed at the boundaries.

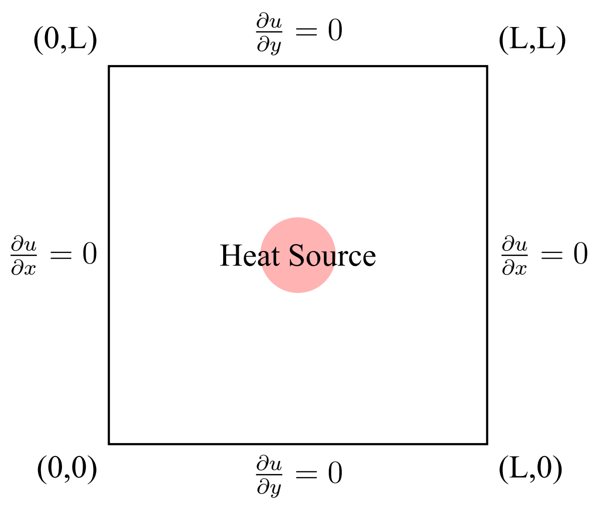

(0,L)
$\frac{\partial u}{\partial y} = 0$
(L,L)
$\frac{\partial u}{\partial x} = 0$
Heat Source
$\frac{\partial u}{\partial x} = 0$
(0,0)
$\frac{\partial u}{\partial y} = 0$
(L,0)

Figure 3.2: Schematic diagram of the square domain showing insulated boundary conditions and centralized heat source.

This assumption is crucial for many real-world applications where thermal insulation is desired, such as in building envelopes, heat exchangers, and electronic packaging, ensuring that heat generated within the domain remains confined without loss to the surroundings. By maintaining these boundary conditions, the model accurately reflects scenarios where external factors do not influence the internal thermal dynamics, allowing for a focused study on the effect of the internal heat source and the material properties governing conduction.

3.1.3 Heat Source

The heat source, a critical component of this study, is strategically positioned at the center of the domain $(0.5, 0.5)$. It is modeled using a Gaussian function to represent the localized energy input which is expressed as

$$f(x, y, t) = S \exp\left(-\frac{(x - x_c)^2 + (y - y_c)^2}{2r^2}\right)$$

[15] In this equation, $S = 500$ indicates the strength of the heat source, effectively quantifying the intensity or rate at which heat is introduced into the system. The coordinates $(x_c, y_c) = (0.5, 0.5)$ specify the central location of the heat source within the domain, aligning with the geometric center of the square. This precise positioning ensures a symmetrical influence on the temperature field, simplifying the analysis while allowing for a clear observation of the heat diffusion pattern. The parameter $r = 0.1$ serves as the characteristic radius of the heat source, defining the spatial extent over which the heat is concentrated. The Gaussian profile chosen for this model offers a smooth and mathematically tractable representation of localized heat input, which is commonly encountered in real-world applications such as laser heating, spot welding, or microchip hotspots. This approach ensures that the heat distribution is realistic and continuous, accurately mimicking how energy from a point source diffuses into the surrounding medium. The Gaussian function's parameters can be adjusted to simulate different source strengths and sizes, providing flexibility and adaptability in modeling various scenarios.

3.1.4 Initial Condition

The initial condition specifies the temperature distribution at the onset of the simulation, establishing a baseline from which the system evolves. In this study, the initial temperature distribution is uniformly set to a room temperature of 25°C across the entire domain i.e

$$u(x, y, 0) = 25°C \quad \forall (x, y) \in \Omega$$

This uniform initial condition signifies that the system begins from a state of thermal equilibrium, with no temperature gradients present before the activation of the heat source. Such a setup is typical in many thermal analyses, providing a controlled environment to observe the effects of a newly introduced heat source. By starting from a uniform temperature, the study can clearly delineate the impact of the localized heating, facilitating an understanding of how the temperature field evolves over time as influenced by both diffusion and the central heat source. This scenario is particularly relevant for applications where systems are initially at rest or in a steady state before experiencing thermal perturbations. The choice of room temperature as the baseline reflects common ambient conditions, making the results applicable and relatable to practical situations. By carefully controlling the initial state, the study isolates the effects of the heat source, providing insights into the fundamental processes governing heat conduction in insulated systems.

3.1.5 PINN Implementation

The implementation of a Physics-Informed Neural Network (PINN) for solving the 2D heat conduction problem with a center heat source is centered around the governing partial differential equation (PDE) which is

$$\frac{\partial u}{\partial t} = \epsilon \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + f(x, y, t)$$

where, $u(x, y, t)$ represents the temperature distribution across the spatial coordinates x and y, and time t. The parameter ϵ is the thermal diffusivity, a material property that quantifies the rate at which heat diffuses through the material. It is a crucial factor in many engineering applications, where the efficiency of heat conduction is a determinant of material performance and safety. The term $f(x, y, t)$ denotes the heat source, capturing the localized energy input that drives the thermal dynamics within the domain. The goal of the PINN is to approximate the solution $u(x, y, t)$ by learning the underlying physics directly from the PDE, initial conditions, and boundary conditions. This approach leverages the expressive power of neural networks while embedding the physical laws governing the system, offering an efficient and flexible alternative to traditional numerical methods, which often involve discretization and can be computationally intensive.

Neural Network Architecture

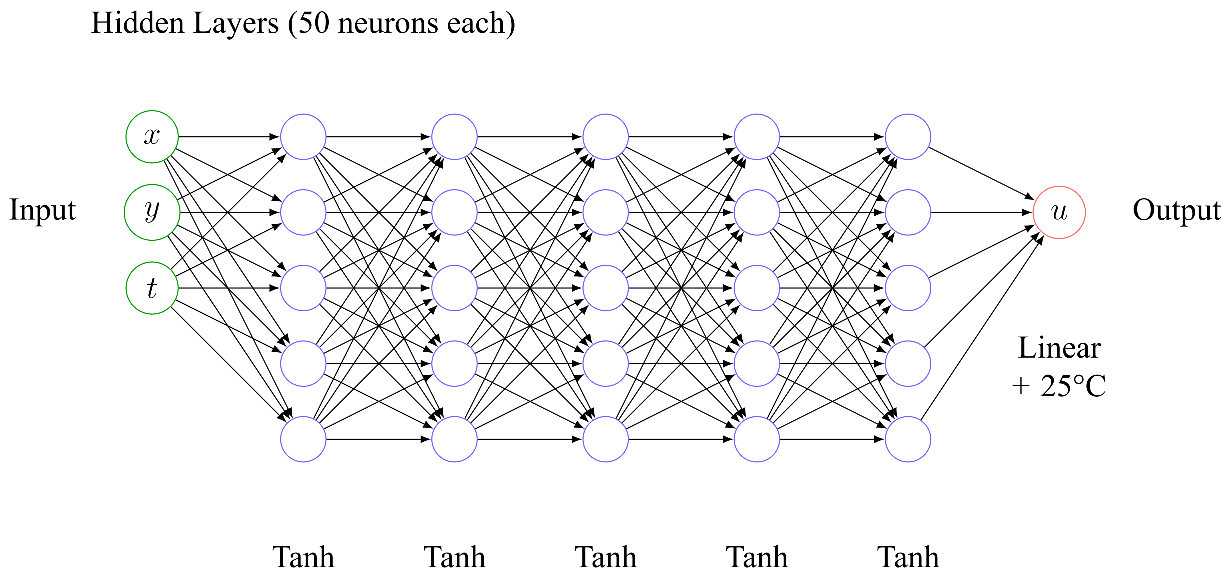


Figure 3.3: Neural network architecture for the 2D heat conduction.

The PINN is implemented using PyTorch, a popular deep learning library, with the following architecture.

```
class PINN(nn.Module):
    def __init__(self, layers, neurons, activation=nn.Tanh()):
        super(PINN, self).__init__()
        self.activation = activation
        self.layers = nn.ModuleList()
        self.layers.append(nn.Linear(3, neurons))
        for _ in range(layers - 1):
            self.layers.append(nn.Linear(neurons, neurons))
        self.layers.append(nn.Linear(neurons, 1))

    def forward(self, x, y, t):
        inputs = torch.cat([x, y, t], dim=1)
        output = inputs
        for layer in self.layers[:-1]:
            output = self.activation(layer(output))
        output = self.layers[-1](output) + 25
        return output
```

The architecture of this Physics-Informed Neural Network (PINN) is specifically designed to approximate the solution $u(x, y, t)$. The key components of this architecture include the input layer, which accommodates three neurons corresponding to the spatial and temporal inputs, x, y, and t. By handling these inputs directly, the network is able to learn the spatial and temporal dependencies inherent in the heat conduction problem, which is crucial for capturing the dynamic behavior of the system as it evolves over time and space.In addition to the input layer, the network comprises five hidden layers, each containing 50 neurons. The number of layers and neurons are hyperparameters that can be adjusted to balance the model's capacity and computational efficiency. While more layers and neurons can increase the network's ability to capture complex patterns, they may also require more data and computational power to train effectively. These hidden layers function as the network's feature extractors, transforming the input data into higher-level representations that facilitate the learning of complex relationships.

The activation function utilized for the hidden layers is the hyperbolic tangent (tanh) function. This choice is motivated by the smoothness and non-linearity of the tanh function, which maps inputs to the range $(-1, 1)$. This property helps in stabilizing gradients during training and provides a smooth approximation of the temperature field. The tanh function's ability to handle varying data distributions makes it suitable for capturing the subtle variations in temperature across the domain.The output layer consists of a single neuron, which outputs the predicted temperature value for the given input coordinates. This scalar output represents the solution to the heat equation at specific points in space and time. By focusing on a single output, the network is streamlined for solving scalar field problems characteristic of many physical systems.

Finally, a constant bias term of 25 is added to the output of the network to initialize predictions around room temperature (25°C). This addition helps to center the network's initial predictions around a realistic baseline, reflecting the initial condition of the system. This bias acts as a corrective measure, anchoring the network's predictions to a known starting condition, which can facilitate faster and more stable convergence during training.The `forward` method defines how input data flows through the network to produce an output, encapsulating the transformations applied at each layer. The use of `torch.cat` concatenates the input tensors, and the loop through `self.layers` applies each layer transformation, followed by the activation function, to propagate the input through the network. This method effectively orchestrates the data processing pipeline, ensuring that each input is systematically transformed into a meaningful prediction.

Loss Function Components

The total loss function for the PINN is constructed as a weighted sum of three distinct components, each enforcing different aspects of the problem's constraints and conditions which can be written as

$$L_{total} = w_1 L_{pde} + w_2 L_{initial} + w_3 L_{boundary}$$

In this implementation, the weights $w_1 = w_2 = w_3 = 1.0$ are chosen to balance the contributions of each loss component, reflecting equal importance in satisfying the PDE, initial conditions, and boundary conditions. These weights can be tuned to emphasize different aspects of the problem, allowing for flexibility in addressing specific challenges or emphasizing certain conditions over others.

3.1.5.1 PDE Residual Loss (L_{pde})

This component of the loss function is crucial as it enforces the satisfaction of the heat equation throughout the domain.

```python
def pde_loss(model, x, y, t, epsilon, f):
    u = model(x, y, t)
    u_t = torch.autograd.grad(u, t, grad_outputs=torch.ones_like(u),
                              create_graph=True)[0]
    u_x = torch.autograd.grad(u, x, grad_outputs=torch.ones_like(u),
                              create_graph=True)[0]
    u_y = torch.autograd.grad(u, y, grad_outputs=torch.ones_like(u),
                              create_graph=True)[0]
    u_xx = torch.autograd.grad(u_x, x, grad_outputs=torch.ones_like(u_x),
                              create_graph=True)[0]
    u_yy = torch.autograd.grad(u_y, y, grad_outputs=torch.ones_like(u_y),
                              create_graph=True)[0]

    residual = u_t - epsilon * (u_xx + u_yy) - f
    return torch.mean(residual ** 2)
```

This function computes the residual of the heat equation using automatic differentiation, which is a key feature of PyTorch that allows for efficient computation of gradients. Automatic differentiation is essential for training PINNs because it enables the precise calculation of all necessary derivatives, facilitating the enforcement of PDE constraints directly in the loss function. Here are the key aspects

- `u_t`, `u_x`, `u_y`: These represent the first-order derivatives of the temperature field with respect to time t, and spatial coordinates x and y, respectively. These derivatives capture the rate of change of temperature in both time and space, which are critical for modeling the dynamic evolution of the system.

- `u_xx`, `u_yy`: These represent the second-order spatial derivatives, providing information about the curvature of the temperature field in the x and y directions. These terms are essential for capturing the diffusion aspect of the heat equation, as they describe how temperature gradients change, leading to heat flow.

- `residual`: This represents the difference between the left and right sides of the PDE. A zero residual indicates that the PDE is perfectly satisfied by the network's predictions.

The residual captures the extent to which the network's predictions deviate from the true physical behavior dictated by the PDE.

- The mean squared error of the residual is returned as the loss, quantifying the extent to which the network's predictions deviate from satisfying the PDE. This loss component is minimized during training, compelling the network to produce solutions that align closely with the physical laws governing the heat conduction process.

This loss component is minimized during training, forcing the network to learn solutions that adhere closely to the physical law described by the heat equation. By accurately capturing the PDE constraints, the network can generalize well to new data points, maintaining consistency with the underlying physics.

3.1.5.2 Initial Condition Loss ($L_{initial}$)

This component ensures that the initial temperature distribution predicted by the network matches the specified initial condition at $t = 0$.

```
def initial_loss(model, x, y, t, u0):
    u = model(x, y, t)
    return torch.mean((u - u0) ** 2)
```

This function computes the mean squared error between the predicted initial temperature and the specified initial temperature $u0$, which is set to 25°C for this problem. By minimizing this loss component, the network is constrained to produce accurate initial conditions, ensuring that the simulation begins from a realistic starting point. The initial condition loss is crucial as it establishes the baseline state of the system, from which all subsequent thermal dynamics evolve. By accurately capturing this initial state, the PINN can more effectively model the temporal evolution of the temperature field. This component ensures that the network respects the initial temperature distribution, providing a solid foundation for the subsequent time-dependent analysis.

3.1.5.3 Boundary Condition Loss ($L_{boundary}$)

In addition to the other components, it's important to enforce the boundary conditions, ensuring that the network's predictions respect the insulation constraints imposed on the domain's edges. Although not fully detailed in the provided code, this component would typically involve computing the mean squared error between the predicted and expected boundary values, similar to the approach used for initial conditions. This would involve using the Neumann boundary condition to ensure no heat flux across the boundaries.

```
def boundary_loss(model, x_boundary, y_boundary, t):
    u = model(x_boundary, y_boundary, t)
    u_n = torch.autograd.grad(u, x_boundary, grad_outputs=torch.ones_like(u
                                           ), create_graph=True)[0]  #
                                           Assuming normal is in x-direction
    return torch.mean(u_n ** 2)
```

This boundary loss function would calculate the first derivative normal to the boundary, ensuring it is zero (no heat flux). Ensuring that the boundary conditions are accurately respected is critical for maintaining the physical integrity of the simulation, particularly in scenarios where insulation plays a key role in system behavior. By accurately capturing these boundary constraints, the PINN ensures that the model's predictions remain consistent with the assumed thermal insulation, preventing unrealistic heat exchanges at the domain edges.

By combining these loss components, the PINN is trained to produce solutions that faithfully represent the physical phenomena described by the heat conduction problem, providing insights into temperature distributions over time and space. The integration of these components into a unified loss function allows the network to balance various physical constraints, leading to a robust solution that generalizes well across the domain.

3.1.5.4 Boundary Condition Loss ($L_{boundary}$)

The boundary condition loss is designed to enforce the Neumann boundary conditions, which represent insulated boundaries where no heat flux occurs across the domain's edges. This condition is mathematically expressed as

$$\frac{\partial u}{\partial n} = 0 \qquad ()$$

where n is the normal vector to the boundary. This condition implies that the derivative of the temperature u normal to the boundary is zero, indicating that there is no change in temperature across the boundary, effectively simulating insulation.

The function defined below implements this loss component.

```
def boundary_loss(model, x, y, t, length):
    u = model(x, y, t)

    x_boundary = (x <= 1e-6) | (x >= length - 1e-6)
    y_boundary = (y <= 1e-6) | (y >= length - 1e-6)

    u_x = torch.autograd.grad(u, x, grad_outputs=torch.ones_like(u),
                                      create_graph=True)[0]
    u_y = torch.autograd.grad(u, y, grad_outputs=torch.ones_like(u),
                                      create_graph=True)[0]

    loss_x = torch.mean(u_x[x_boundary]**2)
    loss_y = torch.mean(u_y[y_boundary]**2)

    return loss_x + loss_y
```

This function enforces the Neumann boundary conditions by computing the gradients of the predicted temperature with respect to the spatial coordinates and penalizing any non-zero values at the boundaries.

- `x_boundary`, `y_boundary`: These are boolean masks used to identify points on the boundaries of the domain. The conditions $x \leq 1e-6$ or $x \geq$ length $- 1e-6$ (similarly for y) effectively capture the edges of the domain, ensuring the boundary conditions are applied correctly.

- `u_x`, `u_y`: These represent the gradients of the temperature field u with respect to x and y. These gradients are computed using automatic differentiation, a powerful feature of PyTorch that allows for efficient calculation of derivatives necessary for enforcing physical constraints.

- The loss penalizes non-zero gradients at the boundaries by calculating the mean squared value of these gradients. Specifically, `loss_x` and `loss_y` are computed as the mean squared gradients at the x and y boundaries, respectively. This penalization ensures that the network's predictions respect the insulated boundary condition, maintaining physical consistency across the domain.

By implementing this loss, the PINN can accurately enforce boundary conditions critical for modeling scenarios where insulation plays a significant role, such as in thermal management systems and insulated enclosures.

3.1.6 Training Process

The training process of the PINN involves several key components, each playing a crucial role in ensuring the network learns an accurate and physically consistent solution to the heat conduction problem.

3.1.6.1 Data Generation

Training points are generated on-the-fly using the `generate_data` function. In the PINN approach, we don't rely on pre-existing datasets. Instead, we generate data points dynamically during the training process. This method allows for adaptive sampling of the problem domain, ensuring a comprehensive exploration of the spatial and temporal dimensions. The data generation process includes creating points for the interior domain, boundaries, and initial conditions, each contributing uniquely to the training process.

This function generates three types of points.

1. **Interior points**: These points are randomly sampled within the domain $[0, L] \times [0, L]$ and time interval $[0, T]$. They are crucial for enforcing the PDE throughout the domain, capturing the bulk behavior of the heat conduction process.

2. **Boundary points**: Sampled on the edges of the domain to enforce boundary conditions. These points are key to ensuring that the predicted solution respects the insulated boundaries, maintaining the physical constraints of the problem. The specific boundary conditions for this heat conduction problem with insulated boundaries are

$$\begin{aligned}
\frac{\partial u}{\partial x}(0, y, t) &= 0 \quad \forall y \in [0, L], t > 0 \\
\frac{\partial u}{\partial x}(L, y, t) &= 0 \quad \forall y \in [0, L], t > 0 \\
\frac{\partial u}{\partial y}(x, 0, t) &= 0 \quad \forall x \in [0, L], t > 0 \\
\frac{\partial u}{\partial y}(x, L, t) &= 0 \quad \forall x \in [0, L], t > 0
\end{aligned}$$

These equations represent Neumann boundary conditions, also known as ”no-flux” or ”insulated” boundary conditions. They ensure that there is no heat flow across the boundaries, which is consistent with insulated edges.

3. **Initial condition points**: Implicitly included by setting $t = 0$ for a subset of the generated points. These points ensure that the initial condition loss is accurately enforced, providing a baseline for the network's predictions.

The number of boundary points is set to 10% of the total number of points, ensuring sufficient coverage of the boundary conditions without overwhelming the training with boundary-specific data. All generated points are assigned the `requires_grad=True` attribute, which is crucial for computing gradients during the training process.

By generating data in this manner, we ensure a good coverage of the problem domain and boundaries, which is essential for the PINN to learn the correct solution to the heat equation. The on-the-fly generation also allows for easy adjustment of the sampling density and distribution as needed during the training process, providing flexibility in addressing specific areas of interest or complexity.

```python
def generate_data(n_points, length, total_time):
    x = torch.rand(n_points, 1, requires_grad=True) * length
    y = torch.rand(n_points, 1, requires_grad=True) * length
    t = torch.rand(n_points, 1, requires_grad=True) * total_time

    n_boundary = n_points // 10
    x_boundary = torch.cat([torch.zeros(n_boundary, 1), torch.full((
                                     n_boundary, 1), length)], dim
                                     =0)
    y_boundary = torch.cat([torch.zeros(n_boundary, 1), torch.full((
                                     n_boundary, 1), length)], dim
                                     =0)
    t_boundary = torch.rand(2 * n_boundary, 1, requires_grad=True) *
                                     total_time

    x = torch.cat([x, x_boundary, torch.rand(2 * n_boundary, 1) *
                                     length], dim=0)
    y = torch.cat([y, torch.rand(2 * n_boundary, 1) * length,
                                     y_boundary], dim=0)
    t = torch.cat([t, t_boundary, t_boundary], dim=0)

    return x.to(device), y.to(device), t.to(device)
```

This function efficiently generates the necessary data points, facilitating the comprehensive training of the PINN. By dynamically adjusting the points based on the current epoch's needs, the network is better equipped to learn complex patterns and behaviors within the domain.

3.1.6.2 Optimizer

: The Adam optimizer is used with a learning rate of 0.001. Adam is a widely used optimization algorithm in deep learning, combining the advantages of two other extensions of stochastic gradient descent, namely AdaGrad and RMSProp. It is well-suited for problems with noisy gradients or when dealing with large datasets and parameters, making it ideal for training neural networks in physics-informed contexts.

```python
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

The choice of optimizer and learning rate significantly impacts the convergence and stability of the training process. Adam's adaptive learning rate adjustments and momentum components help in navigating the loss landscape effectively, reducing the risk of getting stuck in local minima.

3.1.6.3 Training Loop

: The network is trained for 3000 epochs. Each epoch represents a complete pass through the training data, during which the network parameters are updated based on the computed gradients.

```
for epoch in range(epochs):
    x, y, t = generate_data(n_points, length, total_time)
    u0 = torch.full_like(x, 25)  # Initial condition (room temperature)
    f = heat_source(x, y, center_x, center_y, heat_radius,
                                             heat_strength, t)

    optimizer.zero_grad()
    loss_residual = pde_loss(model, x, y, t, epsilon, f)
    loss_initial = initial_loss(model, x, y, torch.zeros_like(t), u0)
    loss_boundary = boundary_loss(model, x, y, t, length)
    loss = weight_residual * loss_residual + weight_initial *
                                             loss_initial +
                                             weight_boundary *
                                             loss_boundary
    loss.backward()
    optimizer.step()
```

In each epoch, new data is generated, losses are computed and backpropagated, and the model parameters are updated. This iterative process allows the network to progressively refine its predictions, learning to satisfy the PDE, initial conditions, and boundary conditions simultaneously. The dynamic generation of data ensures that the network is exposed to a diverse set of scenarios, promoting robust learning and generalization.

3.1.6.4 GPU Acceleration

The implementation leverages GPU acceleration when available. GPUs offer significant computational power, particularly for parallelizable tasks such as matrix operations and gradient calculations, which are common in neural network training.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = PINN(layers, neurons).to(device)
```

By utilizing GPU resources, the training process can be significantly accelerated, enabling the handling of larger datasets and more complex models. This capability is particularly beneficial for physics-informed neural networks, where the computational demands can be substantial due to the integration of complex physical laws and constraints.

3.1.7 Visualization

This visualization step is crucial for interpreting and communicating the results of the trained PINN model. By creating an animated heatmap, we can effectively display the temporal evolution of the temperature distribution across the domain:

- A grid of points is created for visualization, providing a structured representation of the spatial domain. This grid serves as the basis for evaluating the model's predictions and generating the heatmap.

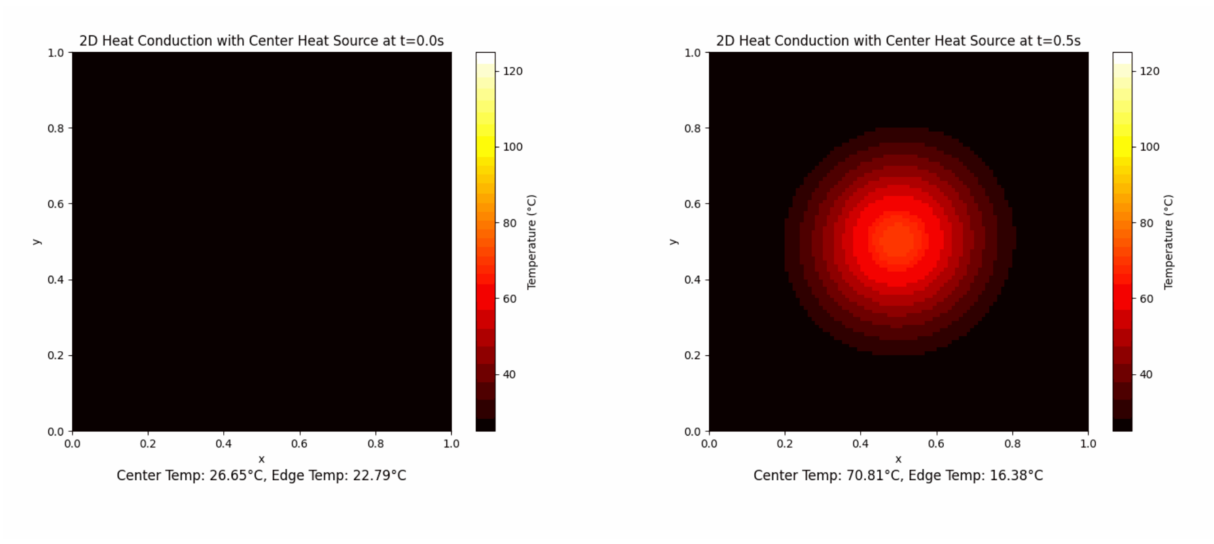
- The update function computes the temperature distribution for each frame of the animation. This function is called repeatedly to update the heatmap with the predicted temperature values at different time steps, creating a dynamic visualization of the heat conduction process.
- The animation shows 51 frames (equivalent to 5 seconds) of the heat conduction process. Each frame corresponds to a specific time step, allowing for a detailed examination of how the temperature field evolves over time.
- Center and edge temperatures are displayed for each frame, providing key metrics that offer insights into the heat distribution dynamics. Monitoring these temperatures helps in assessing the effectiveness of the heat source and the influence of the insulated boundaries on the overall temperature profile.

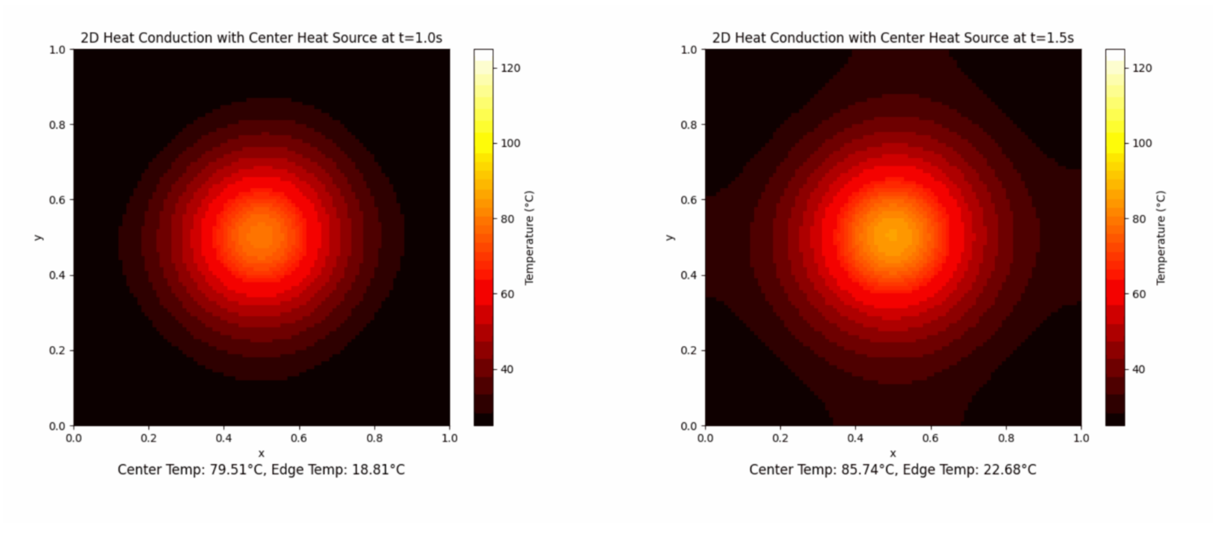
The results are visualized using an animated heatmap created with matplotlib, a widely used plotting library in Python. This visualization not only aids in understanding the model's predictions but also serves as a powerful tool for communicating the results to stakeholders and collaborators.

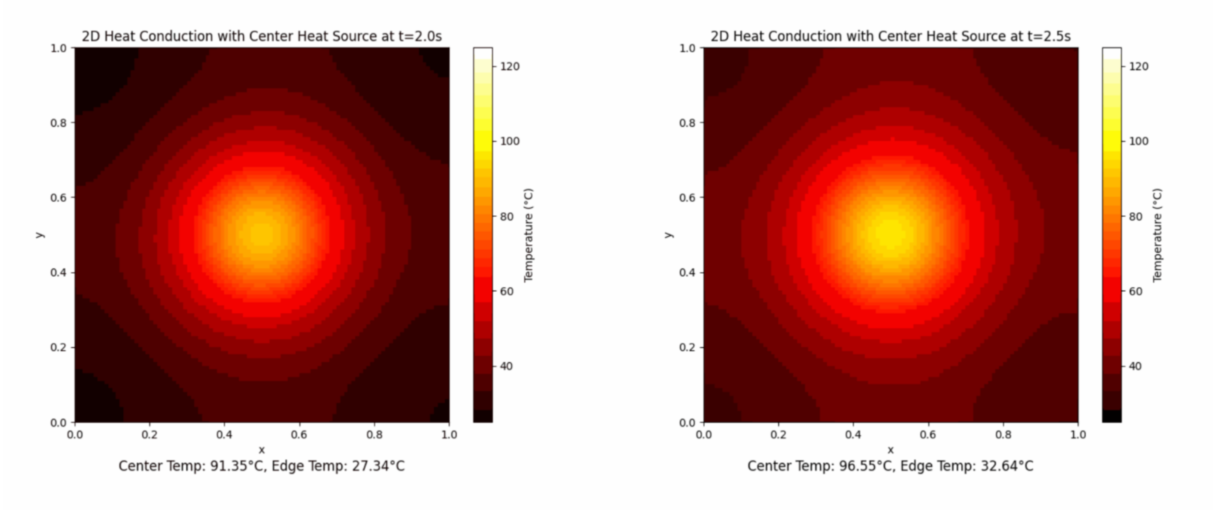
```
fig, ax = plt.subplots(figsize=(8, 7))
x_plot = np.linspace(0, length, n_points_plot)
y_plot = np.linspace(0, length, n_points_plot)
x_plot, y_plot = np.meshgrid(x_plot, y_plot)
x_plot = torch.tensor(x_plot.flatten(), dtype=torch.float32).unsqueeze(1).
                                        to(device)
y_plot = torch.tensor(y_plot.flatten(), dtype=torch.float32).unsqueeze(1).
                                        to(device)

def update(frame):
    t_plot = torch.full_like(x_plot, frame * 0.1).to(device)
    with torch.no_grad():
        u_plot = model(x_plot, y_plot, t_plot).cpu().numpy().reshape(
                                                n_points_plot, n_points_plot)

    im.set_array(u_plot)
    ax.set_title(f'2D Heat Conduction with Center Heat Source at t={frame *
                                            0.1:.1f}s')

    center_temp = u_plot[n_points_plot//2, n_points_plot//2]
    edge_temp = (u_plot[0, 0] + u_plot[0, -1] + u_plot[-1, 0] + u_plot[-1,
                                          -1]) / 4
    temp_text.set_text(f'Center Temp: {center_temp:.2f}\textdegree C, Edge
                                          Temp: {edge_temp:.2f}\textdegree
                                          C')

    return [im, temp_text]

anim = FuncAnimation(fig, update, frames=51, interval=200, blit=True)
anim.save('heat_conduction_center_source_diffusion.gif', writer='pillow',
                                        fps=10)
```

This implementation demonstrates how PINNs can effectively solve the 2D heat conduction problem by incorporating the governing PDE, initial conditions, and boundary conditions directly into the neural network training process. By leveraging advanced visualization techniques, the results are presented in a clear and engaging manner, facilitating a deeper understanding of the complex thermal dynamics at play.

3.1.8 Simulation Results







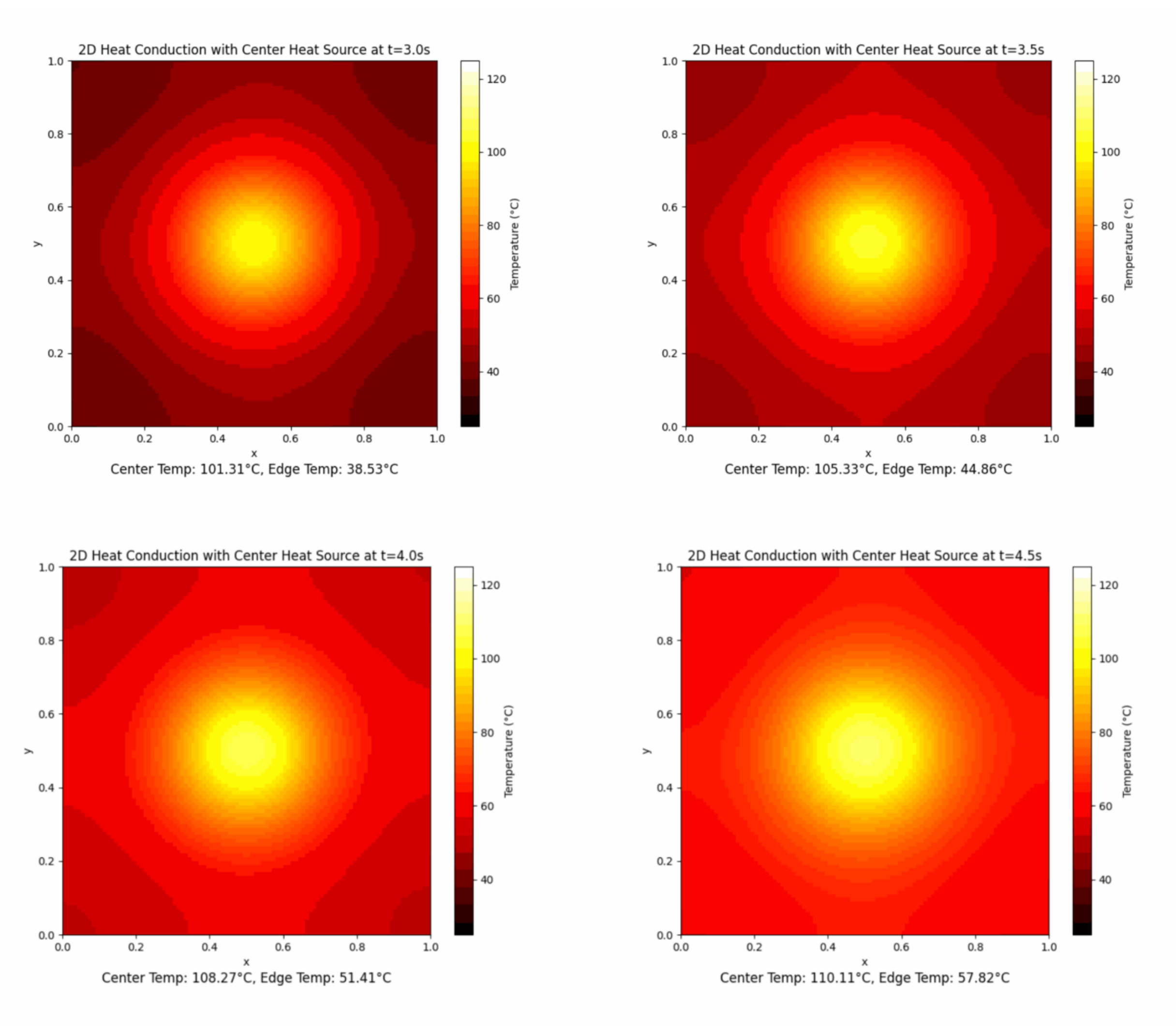


Figure 3.5: Simulations at Different Time Frames: Central heat source

3.1.9 Simulation Analysis

This Physics-Informed Neural Network (PINN) simulation models two-dimensional heat conduction in a square plate, governed by the heat equation which is given by

$$\frac{\partial T}{\partial t} = \alpha \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + Q(x, y, t)$$

In this equation, T represents temperature, t is time, α is the thermal diffusivity (set to 0.01 m²/s in our simulation), and Q is the heat source term. This partial differential equation encapsulates the fundamental physics of heat transfer, describing how thermal energy diffuses through the plate over time. The left-hand side represents the rate of change of temperature at any point, while the right-hand side consists of two components: the spatial diffusion of heat (represented by the second-order partial derivatives) and the additional heat input from the source.

The simulation domain is a 1m x 1m square plate, chosen to represent a standard unit area for ease of analysis. We simulate the heat conduction process for a duration of 5 seconds, providing sufficient time to observe significant temperature changes while keeping computational costs manageable. The initial temperature T_0 is set to 25°C, representing a typical room temperature condition. A central heat source provides continuous thermal energy input, creating a non-uniform temperature distribution that evolves over time.

This setup allows us to investigate how heat propagates from a localized source through a homogeneous material, providing insights into thermal management in various engineering applications, from electronic device cooling to building insulation design.

3.1.9.1 Temperature Evolution Analysis

Center Temperature: We observed a substantial temperature increase at the center of the plate, where the heat source is located. This central point experiences the most direct and intense heating, resulting in the following key observations:

Observation	Value
Initial temperature	$T(0,0,0) = 25°C$
Final temperature	$T(0,0,5) \approx 125°C$
Total temperature rise	$\Delta T \approx 100°C$
Average heating rate	$(\Delta T/\Delta t) \approx 20°C/s$

Table 3.1: Key observations for the center temperature evolution.

The temperature rise at the center follows an approximately exponential curve, which is typical of heating processes approaching equilibrium, represented as

$$T(0,0,t) \approx 25 + A(1 - e^{-kt}),$$

where $A \approx 100°C$ represents the maximum temperature increase, and $k \approx 0.5\ \mathrm{s}^{-1}$ is the heating rate constant. This equation captures the rapid initial heating that gradually slows as the temperature approaches its maximum value. The exponential form arises from the balance between the constant heat input and the increasing rate of heat loss to the surrounding cooler regions as the temperature difference grows.

The high value of k indicates a rapid initial temperature rise, consistent with the localized nature of the heat source. As time progresses, the rate of temperature increase slows, reflected in the exponential decay term. This behavior is crucial in many engineering applications, such as in the design of heat sinks for electronic components, where understanding the rate of temperature rise is essential for preventing thermal damage.

Edge Temperature: The edges of the plate, located 0.5m from the center in each direction, showed a more complex and nuanced temperature evolution.

Observation	Value
Initial temperature	$T(\pm 0.5, \pm 0.5, 0) = 25°C$
Minimum temperature	$T_{min} \approx 24.8°C$ (observed at $t \approx 0.1s$)
Final temperature	$T(\pm 0.5, \pm 0.5, 5) \approx 42°C$
Overall temperature change	$\Delta T \approx 17°C$
Average heating rate (after initial dip)	$\approx 3.5°C/s$

Table 3.2: Key observations for the edge temperature evolution.

Analysis of Temperature Behavior: The edge temperature behavior reveals interesting dynamics in the heat diffusion process, providing insights into the complex interplay of heat transfer mechanisms. Notably, there is an initial temperature dip, with $\Delta T_{dip} = T_{min} - T_0 \approx -0.2°C$. This small but noticeable dip occurs within the first 0.1 seconds of the simulation and may be attributed to several factors: the rapid central heating creates a strong temperature gradient, momentarily drawing heat from the edges towards the center; the finite speed of heat propagation in the material results in a delayed response at the edges; and it could be an artifact of the PINN's initial predictions as the model refines its understanding of the physical system.

After this initial dip, the edge temperature rises steadily, approximated by

$$T(\pm 0.5, \pm 0.5, t) \approx 25 + B(1 - e^{-mt}) \text{ for } t > 0.1s,$$

where $B \approx 17°C$ is the total temperature increase at the edges, and $m \approx 0.3\ \mathrm{s}^{-1}$ is the heating rate constant. This equation describes the gradual heating of the edges as heat diffuses from the center. The lower value of m compared to the center's k reflects the slower heating rate at the edges due to their distance from the heat source. The exponential form of this equation is consistent with the solution to the heat equation for a system approaching thermal equilibrium.

At the end of the simulation ($t = 5s$), the temperature distribution across the plate can be approximated by a two-dimensional Gaussian function which is

$$T(x, y, 5) \approx 42 + 83e^{-(x^2+y^2)/2\sigma^2},$$

where $\sigma \approx 0.3m$ represents the spread of the heat distribution. This equation describes a temperature peak of 125°C at the center, decreasing radially to about 42°C at the edges. The Gaussian shape is characteristic of diffusion processes from a point source, reflecting the spread of heat from the center to the edges of the plate.

This spatial distribution provides key insights: the heat source's effect is most pronounced at the center, creating a localized high-temperature region, while the temperature decreases non-linearly with distance from the center. The rate of decrease is steeper near the center and more gradual towards the edges. The parameter σ quantifies the spread of heat, with approximately 68

Understanding this spatial distribution is crucial for various applications, such as optimizing the placement of components and designing heat sinks in electronic cooling, gaining insights into how different materials conduct and distribute heat in material science, and aiding in planning insulation and HVAC systems for efficient temperature control in building design. The Gaussian approximation, while simplified, captures the essential features of the heat distribution and provides a tractable mathematical model for further analysis and prediction.

3.1.10 Dirichlet Boundary Condition: Central Source Heating

In this variation of the heat conduction problem, we implement Dirichlet boundary conditions while maintaining a central heat source. The key difference lies in how we treat the boundaries of the domain.

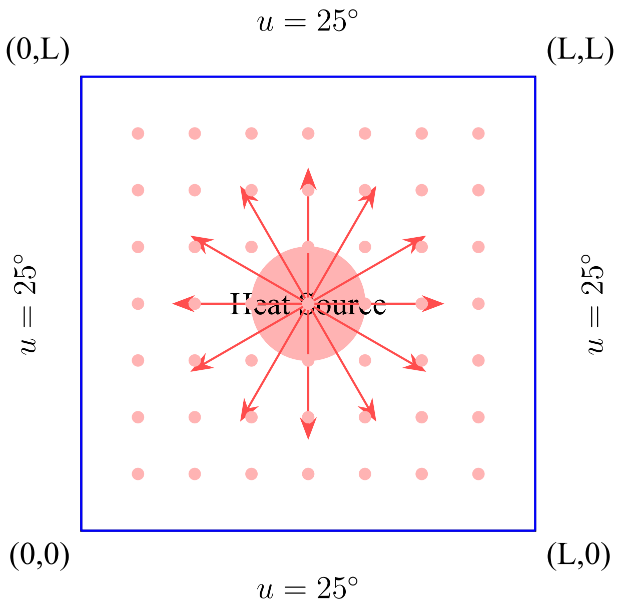


Figure 3.6: 2D Heat Conduction with Center Heat Source and Dirichlet Boundary Condition

3.1.10.1 Boundary Condition

The Dirichlet boundary condition for this problem is expressed as

$$u(x, y, t) = T_b \quad \text{for } (x, y) \in \partial\Omega, t \geq 0$$

where T_b is the fixed boundary temperature (set to 25°C in our implementation), and $\partial\Omega$ represents the boundary of the domain.

3.1.10.2 Mathematical Formulation:

The complete initial-boundary value problem can be stated as

$$\begin{aligned} \frac{\partial u}{\partial t} &= \epsilon \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + f(x, y, t) && \text{in } \Omega \times (0, T] \\ u(x, y, 0) &= T_0 && \text{in } \Omega \\ u(x, y, t) &= T_b && \text{on } \partial\Omega \times [0, T] \end{aligned}$$

where Ω is the spatial domain, T is the final time, T_0 is the initial temperature (room temperature), and $f(x, y, t)$ is the heat source term.

3.1.10.3 Implementation

The Dirichlet boundary condition is enforced through an additional loss term in the PINN as

$$\mathcal{L}_{BC} = \frac{1}{N_b}\sum_{i=1}^{N_b}(u(x_i, y_i, t_i) - T_b)^2$$

where N_b is the number of boundary points, and (x_i, y_i, t_i) are the coordinates of these points.

3.1.10.4 Implementation in Code

The code snippet provided shows how to implement the Dirichlet boundary condition using the function `applyDirichletBoundaryCondition`. This function sets the temperature at each boundary of the 2D temperature array to the specified constant T_D, effectively maintaining that fixed temperature throughout the simulation.

```python
def applyDirichletBoundaryCondition(temperature, T_D=25):
    # Setting the boundary temperatures to T_D
    temperature[0, :] = T_D  # Top boundary
    temperature[-1, :] = T_D  # Bottom boundary
    temperature[:, 0] = T_D  # Left boundary
    temperature[:, -1] = T_D  # Right boundary
```

This implementation ensures that the boundaries maintain a constant temperature, regardless of the internal heat distribution.

3.1.10.5 Physical Interpretation

Unlike the Neumann condition, which models insulated boundaries, the Dirichlet condition represents a scenario where the boundaries are maintained at a constant temperature. This could model, for example, a heating plate surrounded by a temperature-controlled environment.

3.1.10.6 Expected Behavior

With Dirichlet boundary conditions, we anticipate a more rapid dissipation of heat towards the boundaries. And a steady-state solution where the temperature at the center remains elevated due to the continuous heat source, while the edges maintain the fixed boundary temperature.

3.1.10.7 Comparison with Neumann Boundary Condition

The key differences from the Neumann boundary condition implementation are:

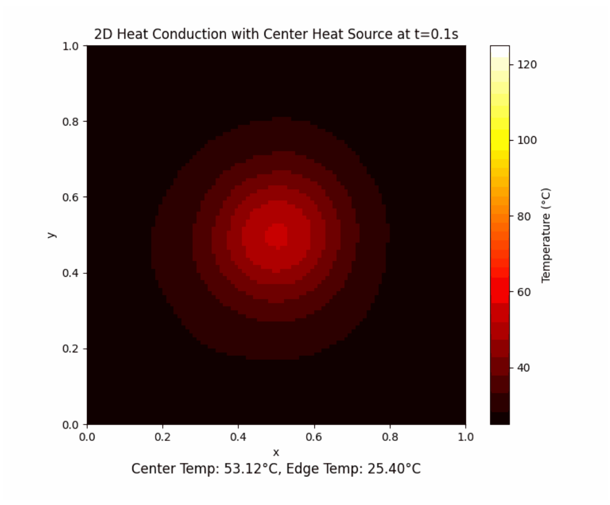
- The boundary temperature is fixed rather than having a zero flux condition.
- The system is not closed in terms of energy, as heat can be lost or gained through the boundaries.
- The long-term behavior will show a balance between the central heat source and the heat loss at the boundaries, rather than an overall temperature increase.

For comparison, the Neumann boundary condition implementation would look like following.

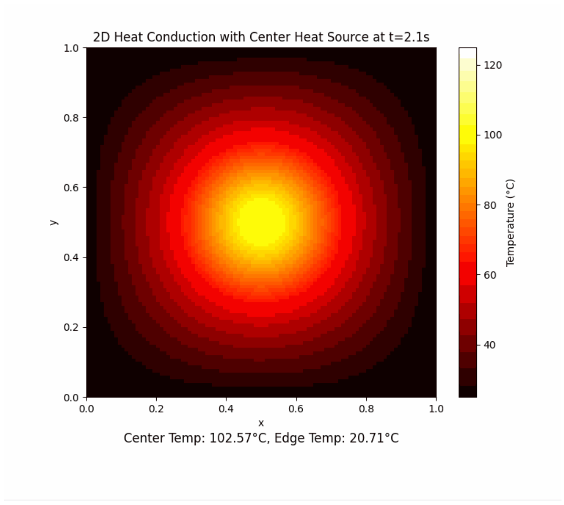
```
def applyNeumannBoundaryCondition(temperature):
    # Assuming temperature is a 2D array
    temperature[0, :] = temperature[1, :]  # Top boundary
    temperature[-1, :] = temperature[-2, :]  # Bottom boundary
    temperature[:, 0] = temperature[:, 1]  # Left boundary
    temperature[:, -1] = temperature[:, -2]  # Right boundary
```

This Neumann implementation sets the temperature gradient to zero at the boundaries, indicating that no heat is lost or gained through the boundary. This results in a fundamentally different behavior compared to the Dirichlet condition. In the Neumann case, heat is conserved within the system, which can potentially lead to an overall temperature increase over time. Conversely, the Dirichlet case allows for heat exchange at the boundaries, maintaining a fixed temperature regardless of internal heat generation. While the Neumann condition might result in more uniform heating in the long term, the Dirichlet condition will preserve a temperature gradient between the center and the edges.This Dirichlet boundary condition implementation provides insights into heat redistribution in systems with fixed-temperature boundaries, offering a contrast to the insulated boundary scenario and demonstrating the PINN's versatility in handling different types of boundary conditions.

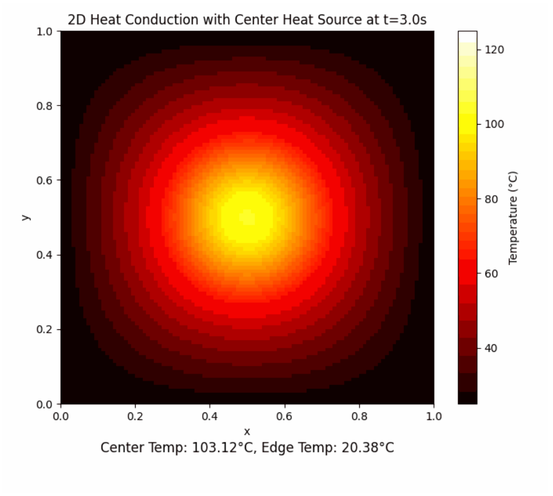
3.1.10.8 Simulation results of Dirichlet boundary condition



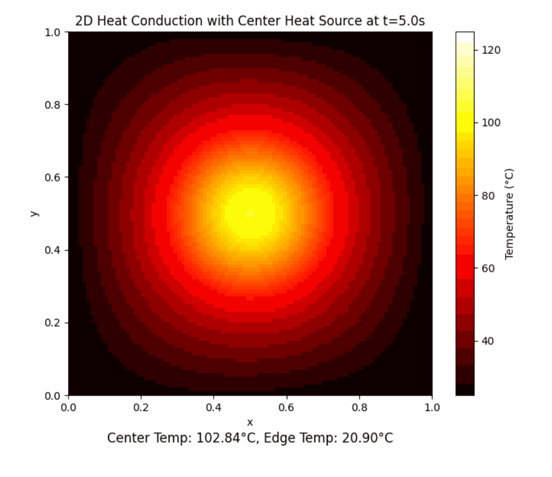
(a) Time 0.1s



(b) Time 2.1s



(c) Time 3.0s



(d) Time 5.0s

Figure 3.7: Simulation: Dirichlet boundary condition

3.1.10.9 Analysis of Dirichlet Boundary Condition Simulation

Simulation Setup

This simulation models 2D heat conduction within a 1m x 1m square plate over a duration of 5 seconds, incorporating a central heat source to simulate localized heating. The framework utilizes a Dirichlet boundary condition to maintain a constant temperature of 25°C along the edges of the plate, ensuring that the boundary does not absorb or dissipate heat during the simulation. This setup allows for the observation of temperature variations over time, particularly as heat diffuses from the center towards the edges.

Temperature Evolution

Table 3.3: Temperature Evolution : Dirichlet boundary condition

Time in [s]	Temp. in Center in $[°C]$	Temp. on Edge in $[°C]$
0.1	53	25
2.1	102	20
3.0	103	20
5.0	102	21

The results indicate that the temperature at the center rises rapidly, reaching a peak of 103°C at around 3.0 seconds. In contrast, the edge temperatures begin at the set boundary condition of 25°C but decline to around 20-21°C by the end of the simulation. This unexpected drop signifies that heat transfer to the edges is not as effective as anticipated, suggesting that the central heating source has a limited impact on the outer regions of the plate.

Spatial Heat Distribution

At $t = 5$ s, a distinct temperature distribution is observed throughout the plate. The central region is characterized by a significantly elevated temperature, peaking at 102°C, which underscores the influence of the central heat source. Conversely, a steep temperature gradient is evident from the center to the edges, indicating a rapid decrease in temperature as one moves outward. The edge temperatures stabilize slightly below the initial 25°C boundary condition—hovering around 21°C. This phenomenon suggests that while the center is effectively heated, the heat does not efficiently propagate to the edges, likely due to the constraints imposed by the boundary conditions.

3.1.10.10 Key Observations

From the simulation results, several notable insights can be drawn. Initially, there is a rapid heating phase at the center, which transitions into a stabilization period with temperatures hovering around 102-103°C after approximately 2 seconds. This rapid increase indicates a swift response of the system to the heat source, leading to a quasi-steady state in the central region. Lastly, the slight deviation of edge temperatures from the set Dirichlet condition raises intriguing questions regarding the heat dynamics at play, suggesting that further investigation into heat transfer mechanisms could provide deeper insights into thermal behavior in similar systems.

3.1.11 Comparison of Neumann and Dirichlet Boundary Conditions

Table 3.4: Comparison of Temperature Behavior

Aspect	**Neumann**	**Dirichlet**
Center Temperature Evolution	Reached approximately 125°C by t = 5s	Plateaued at approximately 102-103°C from t = 2.1s onwards
Edge Temperature	Gradually increased to about 42°C by t = 5s	Slightly decreased to 20-21°C, deviating from the set 25°C
Temperature Gradient (Spatial Distribution)	More gradual gradient from center to edges	Steeper gradient, with a larger temperature difference between center and edges
Heat Spread	Wider spread of the high-temperature region	More concentrated high-temperature region at the center
Rate of Temperature Change	Continuous increase in temperature throughout the simulation	Rapid initial increase followed by stabilization

Key Differences and Their Causes

1. **Boundary Heat Transfer:** Neumann has no heat flux across boundaries, leading to heat accumulation. Dirichlet boundaries act as perfect heat sinks/sources, maintaining a constant temperature.

2. **Energy Balance:** Neumann conserves energy within the system, while Dirichlet allows for energy exchange with the environment through boundaries.

3. **Steady-State Behavior:** Dirichlet reaches steady state more quickly, as excess heat can be removed. Neumann leads to continuous temperature increase.

4. **Spatial Uniformity:** Neumann promotes more uniform temperature distribution over time. Dirichlet maintains stark temperature contrasts between center and edges.

5. **Maximum Temperature:** Dirichlet limits maximum achievable temperature due to constant heat sink at boundaries. Neumann allows for higher peak temperatures.

6. **Edge Behavior:** Unexpected slight cooling at edges in Dirichlet case suggests potential numerical issues or imperfect boundary condition implementation.

3.1.11.1 Implications for Applications

Boundary conditions play a critical role in determining thermal behavior in various systems. Understanding the differences between Dirichlet and Neumann conditions can significantly impact the effectiveness and efficiency of applications. For instance, Dirichlet conditions are ideal for scenarios requiring strict temperature control and localized heating, while Neumann conditions are better suited for insulated systems where natural heat distribution is analyzed. The choice of boundary condition not only affects thermal management but also influences energy efficiency and system stability. Below, we summarize the implications of these boundary conditions for practical applications.

Table 3.5: Implications of Boundary Conditions for Applications

Application Aspect	**Dirichlet Condition**	**Neumann Condition**
Thermal Management	Suitable for strict temperature control	Appropriate for insulated systems or natural heat distribution
Heat Containment	More effective at containing heat near the source	Less effective for localized heating applications
Energy Efficiency	May represent more energy-efficient scenarios	Models scenarios where heat recovery or retention is important
System Stability	Suggests greater thermal stability	May indicate less stability in thermal conditions

3.2 2D Heat Conduction: Redistribution of Heat

This case study investigates the redistribution of heat in a two-dimensional domain using a Physics-Informed Neural Network (PINN). The objective is to model the temporal evolution of temperature distribution within a square region, highlighting the neural network's ability to solve complex partial differential equations (PDEs) that describe heat conduction.

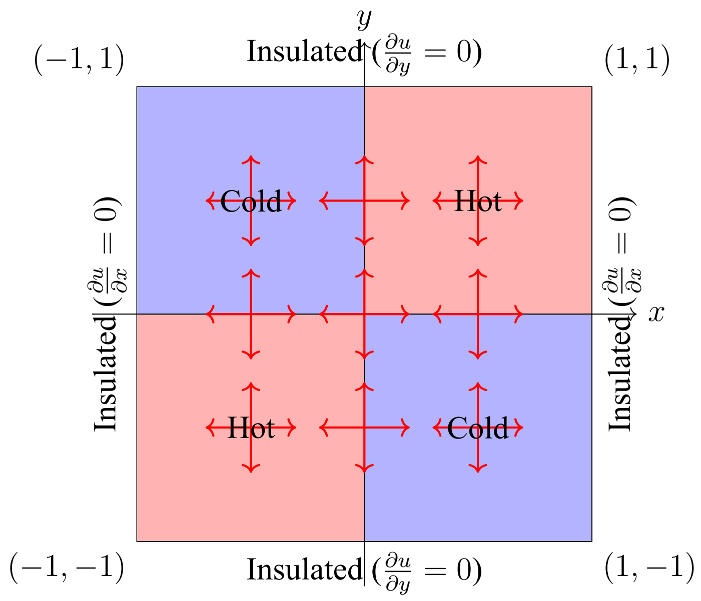

y
x
$(-1,1)$
$(1,1)$
$(-1,-1)$
$(1,-1)$
Insulated ($\frac{\partial u}{\partial y}=0$)
Insulated ($\frac{\partial u}{\partial y}=0$)
Insulated ($\frac{\partial u}{\partial x}=0$)
Insulated ($\frac{\partial u}{\partial x}=0$)
Cold
Hot
Hot
Cold

Figure 3.8: Redistribution of heat with non-uniform initial temperature

3.2.1 Governing Equation

The heat conduction in this study is modeled using the two-dimensional heat equation, which governs the diffusion of heat in a medium. The equation is expressed as

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

where

- $u(x, y, t)$ is the temperature field as a function of spatial coordinates x and y, and time t.
- $\alpha = 0.1$ is the thermal diffusivity, a parameter that influences the rate of heat diffusion.

This PDE is central to the study, as it encapsulates both the temporal and spatial changes in temperature within the domain.

3.2.2 Domain and Boundary Conditions

The study is conducted on a square domain $\Omega = [-1, 1] \times [-1, 1]$, which represents a unit square centered at the origin. The boundaries of this domain are assumed to be perfectly insulated, meaning there is no heat exchange with the environment. Such Neumann boundary conditions are mathematically represented as

$$\frac{\partial u}{\partial n} = 0 \quad \text{on } \partial\Omega$$

where n denotes the outward normal to the boundary $\partial\Omega$.

3.2.3 Initial Condition

The initial temperature distribution within the domain is designed to create a non-uniform starting point for the simulation. The initial condition is defined as

$$u(x, y, 0) = 25 + 25 \cdot \text{sign}(x \cdot y)$$

This configuration results in temperature of 50°C in the upper-right and lower-left quadrants (where x and y have the same sign). And temperature of 0°C in the upper-left and lower-right quadrants (where x and y have opposite signs). And finally a base temperature of 25°C

3.2.4 PINN Implementation

The Physics-Informed Neural Network is implemented to learn the solution to the heat equation while respecting the initial and boundary conditions.

3.2.4.1 Neural Network Architecture

The PINN is implemented using a fully connected neural network with four hidden layers, each containing 32 neurons. The hyperbolic tangent (tanh) activation function is employed to introduce non-linearity.

```python
class PINN(nn.Module):
    def __init__(self, layers):
        super(PINN, self).__init__()
        self.layers = nn.ModuleList()
        self.layers.append(nn.Linear(3, layers[0]))  # Input: (x, y, t)
        for i in range(len(layers) - 1):
            self.layers.append(nn.Linear(layers[i], layers[i + 1]))
        self.layers.append(nn.Linear(layers[-1], 1))  # Output: u(x, y, t)

    def forward(self, x):
        for i in range(len(self.layers) - 1):
            x = torch.tanh(self.layers[i](x))
        x = self.layers[-1](x)
        return x
```

The network takes three inputs (x, y, t) and outputs the predicted temperature u(x, y, t). The use of tanh activation functions allows the network to capture non-linear relationships in the solution.

3.2.4.2 PDE Residual

The PDE residual is computed using automatic differentiation to ensure that the learned solution satisfies the heat equation.

```
def pde_residual(model, x, y, t):
    u = model(torch.cat([x, y, t], dim=1))
    u_t = torch.autograd.grad(u, t, grad_outputs=torch.ones_like(u),
                                    create_graph=True, retain_graph=
                                    True)[0]
    u_x = torch.autograd.grad(u, x, grad_outputs=torch.ones_like(u),
                                    create_graph=True, retain_graph=
                                    True)[0]
    u_xx = torch.autograd.grad(u_x, x, grad_outputs=torch.ones_like(u_x),
                                    create_graph=True, retain_graph=
                                    True)[0]
    u_y = torch.autograd.grad(u, y, grad_outputs=torch.ones_like(u),
                                    create_graph=True, retain_graph=
                                    True)[0]
    u_yy = torch.autograd.grad(u_y, y, grad_outputs=torch.ones_like(u_y),
                                    create_graph=True, retain_graph=
                                    True)[0]
    return u_t - alpha * (u_xx + u_yy)
```

This function computes the first and second-order partial derivatives of u with respect to t, x, and y using PyTorch's autograd functionality. The residual is then calculated as the difference between u_t and α(uxx + u_y), which should be zero for a perfect solution to the heat equation.

3.2.4.3 Loss Function and Training

The training process involves minimizing a composite loss function.

$$\text{Loss} = 10 \times \text{PDE Loss} + \text{IC Loss} + 10 \times \text{BC Loss}$$

where PDE Loss ensures the solution adheres to the heat equation.IC Loss enforces the initial temperature distribution and BC Loss maintains the Neumann boundary conditions.

The network is trained using the Adam optimizer over 20,000 epochs.

```
optimizer = optim.Adam(model.parameters(), lr=1e-4)

for epoch in range(num_epochs):
    optimizer.zero_grad()

    # PDE loss
    pde_loss = torch.mean(pde_residual(model, x, y, t) ** 2)

    # Initial condition loss
    u_pred_ic = model(torch.cat([x_ic, y_ic, t_ic], dim=1))
    ic_loss = torch.mean((u_pred_ic - u_ic) ** 2)

    # Boundary condition loss
    u_left = model(torch.cat([x_bc_left, y_bc, t_bc], dim=1))
    u_right = model(torch.cat([x_bc_right, y_bc, t_bc], dim=1))
    u_bottom = model(torch.cat([x_bc, y_bc_bottom, t_bc], dim=1))
    u_top = model(torch.cat([x_bc, y_bc_top, t_bc], dim=1))

    u_x_left = torch.autograd.grad(u_left, x_bc_left, grad_outputs=torch.
                                    ones_like(u_left), create_graph=
                                    True)[0]
    u_x_right = torch.autograd.grad(u_right, x_bc_right, grad_outputs=torch
                                    .ones_like(u_right), create_graph
```

```
                                              =True)[0]
    u_y_bottom = torch.autograd.grad(u_bottom, y_bc_bottom, grad_outputs=
                                              torch.ones_like(u_bottom),
                                              create_graph=True)[0]
    u_y_top = torch.autograd.grad(u_top, y_bc_top, grad_outputs=torch.
                                              ones_like(u_top), create_graph=
                                              True)[0]

    bc_loss = torch.mean(u_x_left**2 + u_x_right**2 + u_y_bottom**2 +
                                              u_y_top**2)

    # Total loss
    loss = 10 * pde_loss + ic_loss + 10 * bc_loss

    loss.backward(retain_graph=True)
    optimizer.step()
```

The boundary condition loss is computed by evaluating the gradients of u at the boundaries and ensuring they are close to zero, in accordance with the Neumann boundary conditions. The initial condition loss compares the predicted values at t=0 with the specified initial condition.

3.2.5 Visualization and Results

The trained neural network is used to simulate and visualize the temperature distribution over time. An animated GIF is created to illustrate the dynamic redistribution of heat.

```
num_points = 100
num_timesteps = 50
x = torch.linspace(-1, 1, num_points)
y = torch.linspace(-1, 1, num_points)
x, y = torch.meshgrid(x, y, indexing='ij')
x, y = x.reshape(-1, 1), y.reshape(-1, 1)

frames = []
for t in np.linspace(0, 1, num_timesteps):
    t_tensor = torch.ones_like(x) * t
    input_tensor = torch.cat([x, y, t_tensor], dim=1)
    with torch.no_grad():
        u_pred = model(input_tensor).reshape(num_points, num_points).numpy
                                                  ()

    plt.figure(figsize=(10, 8))
    plt.imshow(u_pred, extent=[-1, 1, -1, 1], origin='lower', cmap='hot',
                                              vmin=0, vmax=100)
    plt.colorbar(label='Temperature')
    plt.title(f'Time = {t:.2f}')
    plt.xlabel('x')
    plt.ylabel('y')

    frame_path = os.path.join(save_path, f'frame_{int(t*100):03d}.png')
    plt.savefig(frame_path)
    frames.append(imageio.v2.imread(frame_path))
    plt.close()

imageio.mimsave(gif_path, frames, fps=5)
```

This visualization process creates a series of frames showing the temperature distribution at different time steps. The frames are then combined into an animated GIF, providing a dynamic view of the heat redistribution process over time.

This implementation demonstrates the effectiveness of Physics-Informed Neural Networks in solving PDEs related to heat conduction, highlighting their potential for modeling and understanding thermal processes in various applications.

3.2.6 Analysis of Simulation Results

The analysis of the PINN simulation results for heat redistribution process with Neumann boundary conditions provides insights into the temperature distribution at various time points. As illustrated in Figures 1 through 6, the simulation captures the evolution of temperature from an initial checkerboard pattern to a more uniform distribution over time. The following table summarizes the key observations at random time intervals throughout the simulation.

Table 3.6: Summary of Temperature Distribution Observations

Time (s)	**Observations**
0.00	Initial temperature distribution characterized by a checkerboard pattern. High temperatures (approximately 50°C) in the top-right and bottom-left quadrants; low temperatures (approximately 0°C) in the top-left and bottom-right quadrants. Sharp temperature gradients at interfaces.
0.61	Significant heat diffusion. Sharp boundaries between hot and cold regions begin to blur. Pronounced curving of temperature contours at corners. Overall temperature range narrows, with cold areas warming and hot areas cooling.
1.21	Temperature distribution becomes more uniform. Original checkerboard pattern still discernible but muted. Prominent curved isotherms, particularly near the domain boundaries, reflecting Neumann boundary conditions.
1.82	Further homogenization of the temperature field. Slightly warmer regions persist in the top-right and bottom-left. Near-symmetric temperature distribution consistent with the initial condition and boundary conditions.
2.00	Highly uniform temperature distribution across most of the domain. Subtle temperature variations near the corners indicate incomplete equilibrium. Overall temperature converging towards the average of the initial hot and cold temperatures (approximately 25°C).

3.2.6.1 Key Observations

1. **Rapid Initial Diffusion:** The most dramatic changes occur early in the simulation, consistent with Fourier's law of heat conduction, which predicts faster heat transfer across larger temperature gradients.

2. **Boundary Effects:** The Neumann (zero-flux) boundary conditions are evident in the curved isotherms near the domain edges, demonstrating the conservation of energy within the system.

3. **Symmetry Preservation:** The temperature distribution maintains symmetry throughout the simulation, reflecting the symmetric initial and boundary conditions.

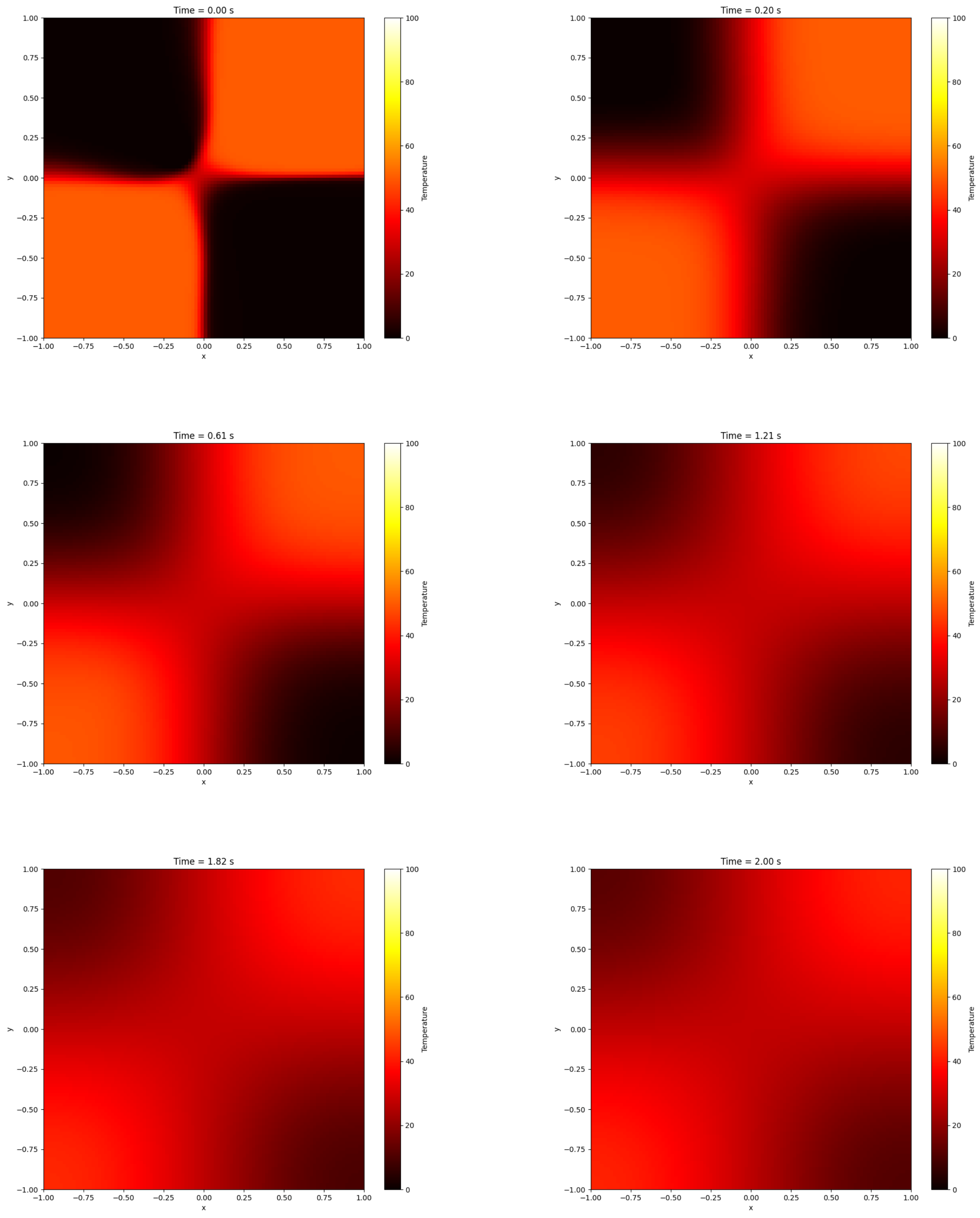


Figure 3.9: Heat redistribution process at different time steps.

4. **Approach to Equilibrium:** While the system has not reached a completely uniform temperature by t = 2 s, it is clearly evolving towards a steady-state condition where the temperature would be uniform across the domain.

These results demonstrate the PINN's capability to capture the complex dynamics of heat diffusion, including the effects of sharp initial gradients and insulated boundary conditions. The simulation successfully models the transition from a highly non-uniform initial state towards thermal equilibrium, showcasing the fundamental principles of heat transfer in a two-dimensional system.

3.3 Uniformly Hot Square Plate with Neumann Boundry

3.3.1 Problem Statement

We consider a two-dimensional heat transfer problem involving a square plate with uniform initial temperature and insulated boundaries. This scenario represents an idealized case of thermal equilibrium and serves as an important benchmark for validating numerical heat transfer simulations.

Let $\Omega = [0, L] \times [0, L]$ be a square domain representing the plate, where L is the side length. The temperature distribution on this plate is denoted by $u(x, y, t)$, where $(x, y) \in \Omega$ and $t \geq 0$ represents time.

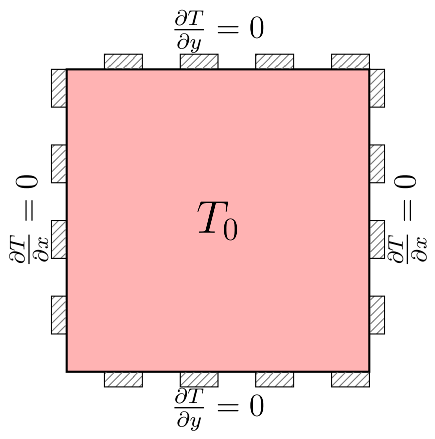

$\frac{\partial T}{\partial y} = 0$
$\frac{\partial T}{\partial x} = 0$
T_0
$\frac{\partial T}{\partial x} = 0$
$\frac{\partial T}{\partial y} = 0$

Figure 3.10: A square plate with uniform initial temperature T_0

The objective is to analyze and simulate the temperature evolution of this system over time, given a uniform initial temperature and Neumann (insulated) boundary conditions.

3.3.2 Mathematical Formulation

The heat conduction in the plate is governed by the two-dimensional heat equation.

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{3.4.2}$$

where

- $u(x, y, t)$ is the temperature at position (x, y) and time t
- α is the thermal diffusivity of the material, defined as $\alpha = \frac{k}{\rho c_p}$
- k is the thermal conductivity
- ρ is the density
- c_p is the specific heat capacity at constant pressure

3.3.3 Initial Condition

At $t = 0$, the temperature is uniform across the entire plate

$$u(x, y, 0) = T_0 \quad \forall (x, y) \in \Omega$$

where T_0 is the initial temperature.

3.3.4 Boundary Conditions

Neumann boundary conditions are applied on all edges of the square, representing perfect insulation which can be represented as

$$\begin{aligned}
\frac{\partial u}{\partial x}(0, y, t) = 0 \quad \forall y \in [0, L], t > 0 \\
\frac{\partial u}{\partial x}(L, y, t) = 0 \quad \forall y \in [0, L], t > 0 \\
\frac{\partial u}{\partial y}(x, 0, t) = 0 \quad \forall x \in [0, L], t > 0 \\
\frac{\partial u}{\partial y}(x, L, t) = 0 \quad \forall x \in [0, L], t > 0
\end{aligned} \tag{3.4.4}$$

These conditions ensure that there is no heat flux across the boundaries of the plate.

3.3.5 Analytical Solution

In this particular case, we can derive an analytical solution to the problem. Let's consider the implications of our initial and boundary conditions.

Given the uniform initial temperature $T_0 = 61.30$ °C, we have

$$\nabla^2 u(x, y, 0) = \frac{\partial^2 u}{\partial x^2}(x, y, 0) + \frac{\partial^2 u}{\partial y^2}(x, y, 0) = 0 \quad \forall (x, y) \in \Omega \tag{3.4.5}$$

This is because the temperature is constant across the domain, so all spatial derivatives are zero.

The Neumann boundary conditions (Eq. 3.4.4) ensure that there is no temperature gradient at the boundaries. This means that the heat flux across the boundaries is zero,

$$-k\nabla u \cdot \mathbf{n} = 0 \quad \text{on } \partial\Omega$$

where **n** is the outward normal vector to the boundary $\partial\Omega$.

Substituting Eq. 3.4.5 into the heat equation (Eq. 3.4.2), we get

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u = \alpha \cdot 0 = 0$$

This implies that $\frac{\partial u}{\partial t} = 0$ for all $(x, y) \in \Omega$ and $t > 0$.

Therefore, the analytical solution to this problem is

$$u(x, y, t) = 61.30\,°\text{C} \quad \forall (x, y) \in \Omega, t \geq 0$$

This solution satisfies the heat equation, the initial condition, and the boundary conditions.

3.3.5.1 Physical Interpretation

The constant temperature solution can be explained by the following physical principles:

1. **Conservation of Energy:** With insulated boundaries, there is no mechanism for the system to exchange heat with its surroundings. Therefore, the total thermal energy of the system must remain constant.
2. **Second Law of Thermodynamics:** Heat naturally flows from regions of higher temperature to regions of lower temperature. In this case, there are no temperature gradients within the system to drive heat flow.
3. **Thermal Equilibrium:** The system starts in a state of thermal equilibrium (uniform temperature) and, with no external influences, remains in this equilibrium state.

3.3.6 Numerical Simulation

Despite the trivial nature of the analytical solution, it is instructive to perform a numerical simulation of this scenario. This serves several purposes like validation of the numerical method and verification of boundary condition implementation.

For our simulation, we use the following parameters:

Domain: $[0, 1] \times [0, 1]$

Initial temperature: 61.30°C

Thermal diffusivity: 0.1 m^2/s

Simulation time: $[0, 1]$

Spatial discretization: 100 $\times$ 100 grid

Time step: 0.001 s

We employ a finite difference method with forward Euler time integration as

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \alpha \left(\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{(\Delta x)^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{(\Delta y)^2} \right)$$

[16]

where $u_{i,j}^{n}$ represents the temperature at grid point (i, j) and time step n.

The Neumann boundary conditions are implemented using ghost points as

$$
\begin{aligned}
u^n_{-1,j} &= u^n_{1,j} \quad \text{(left boundary)} \\
u^n_{N_x+1,j} &= u^n_{N_x-1,j} \quad \text{(right boundary)} \\
u^n_{i,-1} &= u^n_{i,1} \quad \text{(bottom boundary)} \\
u^n_{i,N_y+1} &= u^n_{i,N_y-1} \quad \text{(top boundary)}
\end{aligned}
\tag{3.4.6}
$$

where N_x and N_y are the number of grid points in the x and y directions, respectively.

3.3.7 Simulation Results

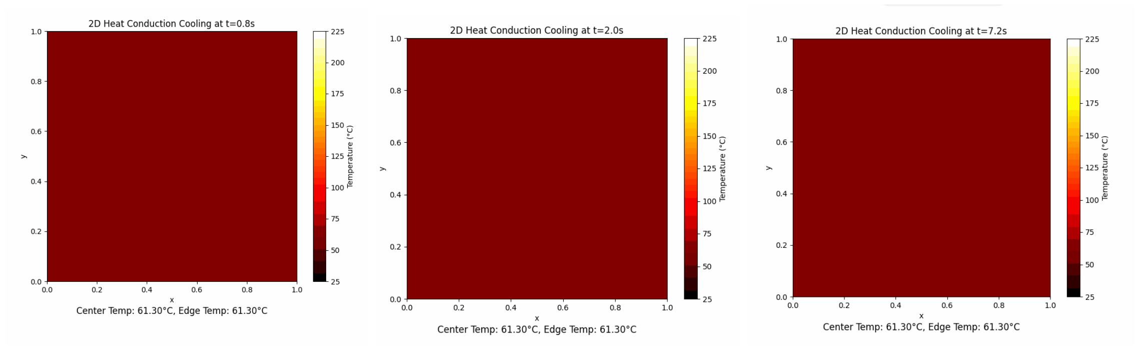


Figure 3.11: Temperature distribution Uniformly Hot Square Plate

Figure 3.11 shows the temperature distribution at three different random time steps: $t = 0.8$, $t = 2.0$, and $t = 7.2$. As predicted by the analytical solution, the temperature remains constant at 61.30°C and uniform throughout the simulation.

3.3.8 Error Analysis

To quantify the accuracy of our numerical simulation, we compute the maximum absolute error between the numerical solution and the analytical solution which is expressed as

$$E_{\text{max}} = \max_{i,j} |u^n_{i,j} - 61.30|$$

For our simulation, we find that $E_{\text{max}} < 10^{-12}$ for all time steps, which is within the expected range of floating-point precision errors.

3.3.9 Discussion

The simulation of a uniformly hot square plate with Neumann boundary conditions illustrates several important concepts in heat transfer and numerical methods:

1. **Importance of Boundary Conditions:** This case demonstrates how boundary conditions fundamentally determine the behavior of a heat transfer system. The insulated boundaries prevent any heat exchange with the environment, leading to a constant temperature solution.

2. **Equilibrium States:** The system begins and remains in thermal equilibrium. This highlights the concept that without external influences or internal gradients, a system will maintain its equilibrium state.

3. **Numerical Method Validation:** While the solution is trivial, this case serves as an excellent test for numerical methods. It verifies that the method correctly implements Neumann boundary conditions and maintains a constant solution when appropriate.

4. **Conservation Properties:** The simulation demonstrates the conservation of energy in the system. The total thermal energy remains constant throughout the simulation, as expected for an insulated system.

5. **Stability and Accuracy:** The numerical solution remains stable and accurate over long simulation times, which is crucial for more complex heat transfer problems.

The study of a uniformly hot square plate with Neumann boundary conditions provides valuable insights into heat transfer processes and numerical simulation techniques. While the constant temperature solution might seem trivial, it serves as a fundamental baseline case that validates the correct implementation of Neumann boundary conditions and demonstrates the ability of numerical methods to maintain constant solutions. Moreover, it illustrates the importance of temperature gradients or external influences in driving heat transfer. This scenario also serves as a simple test case for assessing the conservation properties and long-term stability of numerical schemes. Understanding and correctly simulating such fundamental cases is crucial for building confidence in more complex heat transfer simulations, as it lays the groundwork for tackling more challenging problems involving non-uniform initial conditions, mixed boundary conditions, or additional heat sources and sinks.

4 Conclusion

This thesis presented a comprehensive study on the simulation of two-dimensional heat conduction using Physics-Informed Neural Networks (PINNs). By integrating the principles of heat conduction with advanced neural network methodologies, we demonstrated the potential of PINNs as an effective tool for solving complex thermal problems. The initial chapters provided a foundational understanding of heat conduction principles and neural network architectures, setting the stage for the application of PINNs. Through various case studies, including a central heat source setup, heat redistribution scenarios, and the behavior of a uniformly hot square plate, we highlighted the adaptability of PINNs in modeling heat conduction phenomena under different boundary conditions.

The simulation results showed that PINNs could accurately capture temperature distributions and dynamics, often outperforming traditional numerical methods such as Finite Difference Method (FDM) and Finite Element Method (FEM). This was particularly evident in scenarios with complex geometries and boundary conditions, where PINNs provided a more flexible and efficient approach.Despite the promising results, this study also acknowledged the limitations of PINNs, including the need for extensive training data and the challenges associated with convergence in certain configurations. These insights pave the way for further research in optimizing the training process and enhancing the robustness of PINNs in thermal simulations.

Looking ahead, future work could explore the integration of PINNs with other machine learning techniques to refine predictions and improve computational efficiency. Additionally, expanding the application of PINNs to three-dimensional heat conduction problems and real-world engineering scenarios could significantly advance the field.In summary, this research underscores the transformative potential of Physics-Informed Neural Networks in thermal simulations, offering a novel perspective on the intersection of physics and artificial intelligence.

The implementation of the algorithms and the associated code were developed using Python and various libraries. All the code developed and utilized for this thesis is available in my personal GitHub repository [17]. This repository contains all relevant scripts, documentation, and additional resources that support the findings presented in this thesis.

Bibliography

[1] Elsevier, *Thermal conduction*, Accessed: [10.08.2024]. [Online]. Available: https://booksite.elsevier.com/samplechapters/9780123735881/9780123735881.pdf.

[2] Wikipedia, *Thermal conduction*, Accessed: [10.08.2024]. [Online]. Available: https://en.wikipedia.org/wiki/Thermal_conduction.

[3] T. L. Bergman, A. S. Lavine, F. P. Incropera, and D. P. DeWitt, *Introduction to heat transfer*, Accessed: [15.08.224]. [Online]. Available: https://ia601300.us.archive.org/5/items/bzbzbzHeatTrans/Heat%20and%20Mass%20Transfer/Bergman%2C%20Incropera/Introduction%20to%20Heat%20Transfer%206e%20c.2011%20-%20Bergman%2C%20Incropera.pdf.

[4] webbusterz, *Thermal conduction*, Accessed: [10.08.2024]. [Online]. Available: https://www.webbusterz.org/understanding-heat-transfer/.

[5] S. J. Mohaiminul Islam, *Neural networks*, Accessed: [31.08.2024]. [Online]. Available: https://www.researchgate.net/publication/337137421_An_Overview_of_Neural_Network.

[6] Y. Goldberg, "A primer on neural network models for natural language processing," Oct. 2016. [Online]. Available: https://arxiv.org/abs/1510.00726.

[7] B. P. C, *Regularization in neural networkss*, Accessed: [10.10.2024]. [Online]. Available: https://www.pinecone.io/learn/regularization-in-neural-networks/.

[8] Wikipedia, *Neural networks*, Accessed: [10.08.2024]. [Online]. Available: https://en.wikipedia.org/wiki/Neural_network.

[9] B. Moseley, "So, what is a physics-informed neural network?," 2023. [Online]. Available: https://benmoseley.blog/my-research/so-what-is-a-physics-informed-neural-network/ (visited on 08/31/2023).

[10] H. Song, "2d heat conduction applications," Mar. 2017. DOI: https://www.sciencedirect.com/science/article/pii/S2542435118300345.

[11] M. Adam, "Solving 1d and 2d heat conduction equations and advection equation," Mar. 2023. DOI: 10.13140/RG.2.2.11783.98720.

[12] A. Chandra and R. Kumar, "Finite difference method and computational fluid dynamics," Mar. 2022. [Online]. Available: https://www.jafmonline.net/article_1282_37dc9e7bbfb530c05996c31b03ce8774.pdf.

[13] Mathoverflow, "Physical interpretation of robin boundary conditions," May 2020. [Online]. Available: https://mathoverflow.net/questions/95316/physical-interpretation-of-robin-boundary-conditions.

[14] R. A. Bafghi, *Pytorch*, Accessed: [31.08.2024]. [Online]. Available: https://www.researchgate.net/publication/376521877_PINNs-Torch_Enhancing_Speed_and_Usability_of_Physics-Informed_Neural_Networks_with_PyTorch.

[15] Wikipedia, *Gaussian function*, Accessed: [11.08.2024]. [Online]. Available: https://en.wikipedia.org/wiki/Gaussian_function.

[16] Wikipedia, *Euler method*, Accessed: [10.10.2024]. [Online]. Available: https://en.wikipedia.org/wiki/Euler_method.

[17] S. Aryal, *Bachelor thesis: Pinns for 2d heat conduction*, Accessed: 2024-10-28. [Online]. Available: https://github.com/Samman2571/Bachelor_thesis_PINNs_2D_Heat_Conduction.

[18] R. A. Bafghi, *Pytorch*, Accessed: [31.08.2024]. [Online]. Available: https://www.researchgate.net/publication/380824296_Comparing_PINNs_Across_Frameworks_JAX_TensorFlow_and_PyTorch.

[19] G. Karniadakis, Y. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, pp. 1–19, May 2021. DOI: 10.1038/s42254-021-00314-5.

[20] M. Raissi, *Pinns*, Accessed: [10.04.2024], 2023. [Online]. Available: https://github.com/maziarraissi/PINNs.

[21] C. AB, *Pinn heat equation*, Accessed: [10.04.2024], 2023. [Online]. Available: https://github.com/cissieAB/pinn-heat-equation.

[22] 314arhaam, *Heat pinn*, Accessed: [10.04.2024], 2023. [Online]. Available: https://github.com/314arhaam/heat-pinn.