HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Parameter Identification of Controlled Heat Conduction

Bachelor Thesis

E-mobility and Green Energy (B.Eng)
Ravensburg-Weingarten University of Applied Sciences

Harshit Isamaliya
Matriculation Number: 33468

Prof. Dr.-Ing. Lothar Berger (First Reviewer)
Herr Stephan Scholz (M.Sc.) (Second Reviewer)

September 2024

Abstract

This thesis focuses on investigating the identification of parameters in controlled heat conduction within one-dimensional rods. The main focus is on estimating thermal conductivity (λ) and controller gain (K_p) through the utilization of open-loop and closed-loop systems. A proportional controller (P-Controller) is utilized in the closed-loop setup to achieve target temperatures.

The methodology involves solving the forward problem to model heat conduction and solving the inverse problem to estimate the parameters. Three optimization techniques, specifically Nelder-Mead, Conjugate Gradient, and Newton's Method, are utilized for this purpose. To address system noise, Gaussian noise is introduced to replicate real-world industrial conditions, contrasting with noiseless, idealized simulations. The study compares various sensor configurations to determine the necessary number of sensors to accurately determine , particularly in the presence of noise.

The findings indicate that increasing the number of sensors and implementing suitable optimization techniques significantly improves estimation accuracy in noisy environments. This study provides valuable insights into sensor placement and optimization techniques, making the proposed methodology highly applicable for improving the accuracy and efficiency of industrial heat conduction systems.

Contents

List of Tables

List of Figures

1 Introduction

Heat transfer through conduction is a critical phenomenon in engineering and applied physics. It plays a key role in numerous technological processes, from small-scale microelectronic devices to large industrial systems. The accurate control and prediction of thermal energy transfer is essential for maximizing performance, ensuring operational safety, and improving energy efficiency [1, 2]. Successful thermal control relies on precisely defining the factors that impact heat transfer, a challenging task due to the complexities of real-world systems [3].

Heat conduction, the transfer of thermal energy between neighboring atoms and molecules, is driven by temperature differences. This process involves particles with more kinetic energy transferring their energy to particles with lower kinetic energy, resulting in heat flow from regions of higher to lower temperatures [4]. However, in practical applications, this process is complicated by material properties, boundary conditions, and external controls [5]. Understanding how these elements influence heat conduction dynamics is crucial for developing effective thermal management systems.

Measuring the parameters that control heat conduction directly is often challenging due to cost and practical limitations. To address these difficulties, techniques for identifying parameters provide a solution by estimating system variables that are unknown to measure based on measurable data, such as temperature distributions over time. These techniques enable the resolution of inverse problems, leading to more accurate modeling and control of heat conduction in various applications [6].

This thesis is specifically focused on the identification of two unknown parameters within a controlled heat conduction system: the thermal conductivity (λ)of the material and the gain (K_p) of a proportional controller used to regulate the system temperature. The system under study is a one-dimensional rod with a heating element at one end and distributed temperature sensors along its length. A primary sensor at the opposite end of the rod monitors the target temperature, and the P-Controller adjusts the heat input to reach this target. The challenge lies in identifying λ and K_p using inverse methods, both in open-loop and closed-loop scenarios, with the latter involving active temperature regulation through feedback control.

This study's crucial focus is to maximize parameter estimation accuracy by determining the best number and spatial arrangement of sensors. The quality of the estimated parameters is directly influenced by the placement and density of sensors, requiring a trade-off between computational efficiency and estimation precision [7]. Various optimization techniques, including the Nelder-Mead Method, Conjugate Gradient Method, and Newton's Method, are employed in this study, along with automatic differentiation to enhance gradient calculations. This is especially crucial in the closed-loop scenario, where the dynamics of the P-Controller add complexity to the inverse problem.

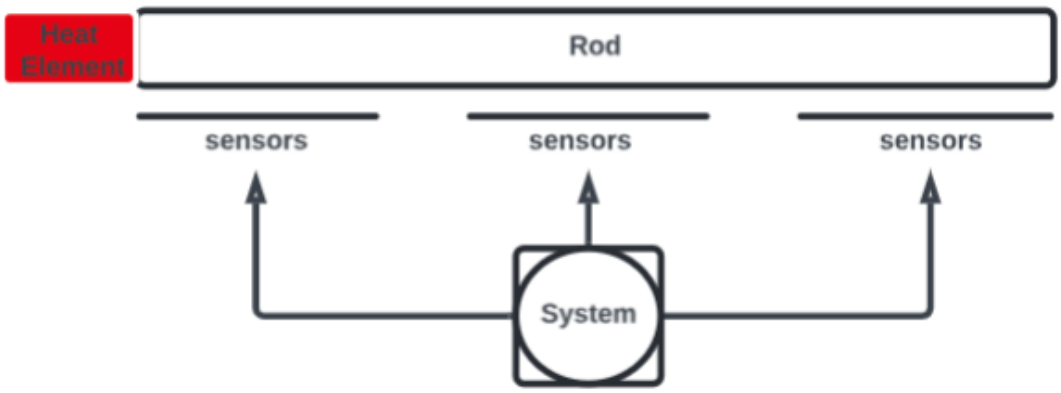

Figure 1.1: Schematic Representation of Sensor Placement in a Controlled Heat Conduction System

The study extensively examines identifying parameters in controlled heat conduction systems by integrating theoretical modeling with practical considerations. In addition to tackling the technical obstacles, it considers sensor noise and time delays to more accurately reflect real-world situations. The results have implications for industries where precise temperature regulation is critical, such as industrial heat management and climate control systems. Ultimately, this investigation enhances the understanding of parameter identification and offers valuable insights into improving sensor placement for increased accuracy in controlled heat conduction systems.

Mathematical Model and Problem Formulation

2

The following chapter will create a mathematical framework to depict heat transfer through a rod. We heat the rod from the left-hand side and mount temperature sensors at many locations along its length in order to measure the flow of heat. We will set up the problem, convert it into a form suitable for numerical solutions, and specify the initial and boundary conditions.

2.1 Description of the Physical System

The problem is set in a physical system consisting of a metallic rod of length (L), which starts at a temperature of 300 K (Kelvin). The system is shown in Fig.2.1. The heating element is located at x_1 on the left side of the rod. The heat propagates along the rod from the heating source towards the insulated end. The idea is to take a look at how the heat diffuses, using sensors across the rod from x_1 through x_{20}.

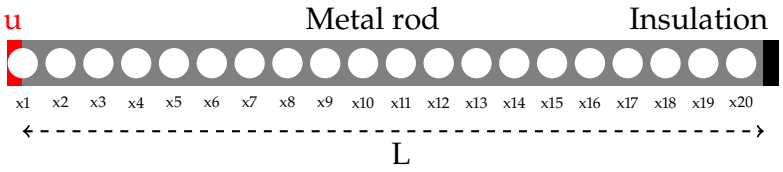

Figure 2.1: Schematic diagram

The heating element introduces energy into the system, causing the temperature at each point on the rod to change over time. To understand and predict this temperature distribution, we implement a mathematical model called the heat equation. This model explains the movement of heat from hotter to cooler regions of the rod.

2.1.1 Heat Equation

The heat equation, a Parabolic partial differential equation (PDE), describes the heat distribution in a given region over time [8]. For a one-dimension rod, the heat equation simplifies,

$$\frac{\partial T(x,t)}{\partial t} = \frac{\lambda}{\rho \cdot c} \frac{\partial^2 T(x,t)}{\partial x^2}. \tag{2.1}$$

This equation models the temperature distribution in a rod over time, where $T(x,t)$ represents the temperature at position "x" along the rod at time "t". The left-hand side of the equation, $\frac{\partial T(x,t)}{\partial t}$, is the partial derivative of the temperature concerning time, representing the temperature change rate over time at a given position.

On the right-hand side of the equation, λ represents the material's thermal conductivity, which quantifies how well the material conducts heat. The symbols ρ and c refer to the material's density and specific heat capacity, respectively. The product $\rho \cdot c$ is the volumetric heat capacity, indicating the material's ability to store heat.

The second spatial derivative of the temperature, $\frac{\partial^2 T}{\partial x^2}$, represents the curvature of the temperature profile along the rod. The concept is essential for our analysis as it shows how the temperature at a specific location varies from its surrounding positions.

The temperature evolution in a rod is elegantly described by the heat equation, which establishes a relationship between the temperature's rate of change over time and the heat diffusion throughout the rod. The thermal diffusivity, represented by the parameter α (i.e., the ratio of $\frac{\lambda}{\rho \cdot c}$), is vital in determining how rapidly heat propagates within the material.

However, defining the conditions at the rod's boundaries is imperative to solve this equation entirely and accurately. These boundary conditions offer essential information for determining whether heat can flow into or out of the system.

2.1.2 Neumann boundary condition

The boundary conditions determine how heat flow behaves at the boundaries. In this particular case, the right side of the rod is insulated, preventing heat from escaping through that boundary. This insulation is indicated by a Neumann boundary condition, which states that the heat flux at the boundary is zero [9],

$$\left.\frac{\partial T}{\partial x}\right|_{boundary} = q.$$

For an insulated boundary, when $q = 0$, no heat leaves the system at this boundary, which reflects the physical condition of insulation and enables a complete description of the temperature distribution along the rod. In order to solve the heat equation and monitor temperature changes using numerical methods, we need to represent the continuous spatial domain by dividing the rod into discrete points where temperature values can be computed.

2.2 Spatial Discretization

To solve the heat equation numerically, we partition the spatial domain into smaller components. The rod is discretized into N_x equally distant points, which represent the positions of sensors for calculating temperature values. By dividing the rod into discrete intervals, we can estimate the temperature distribution and numerical techniques to predict heat transfer over time.

2.2.1 Discretization Approach

The rod is divided into N_x evenly spaced points when we discretize it. The distance between adjacent points is denoted as Δx, which is calculated as

$$\Delta x = \frac{L}{N_x - 1}.$$

The rod's total length is denoted as L, and the number of points, including those at both ends of the rod, is represented by N_x. This guarantees that the rod is divided into $N_x - 1$ intervals of equal length Δx [10].

2.2.2 Central Difference Method

After dividing the rod into points, we must find a method to estimate the spatial changes in temperature at each point in the rod, as specified by the heat equation. A commonly used method for this is the Central Difference Method, which involves using the temperature values at adjacent points to compute an approximation of the derivative.

For a spatial grid with equally spaced points x_0,x_1,x_3,....,x_N, the second derivative of temperature concerning space at the point x_i is approximated by,

$$\frac{\partial^2 T_n}{\partial x^2} \approx \frac{T_{n-1} - 2T_n + T_{n+1}}{\Delta x^2}.$$

In this case, the temperatures at neighboring points, T_{n-1}, T_n, T_{n+1}, are used to estimate the curvature of the temperature profile through the central difference approximation, which makes use of the temperature at the points on either side of T_n.

2.2.3 Semi-Discrete Heat Equation

By substituting this central difference approximation into the heat equation, we can derive the semi-discrete heat equation. The given equation describes how the temperature changes at every point inside the rod.

$$\frac{dT_n}{dt} = \alpha \frac{T_{n-1} - 2T_n + T_{n+1}}{\Delta x^2}.$$

The spatial domain has been divided into points, leaving time as a continuous variable. Hence, this method is called '**Semi-Discrete**.' Consequently, there is a set of Ordinary Differential Equations (ODEs) in the temperature evolution at each spatial point in the rod. To thoroughly explain the transfer of heat along the rod, it is important to establish the characteristics of heat at the boundaries. These ODEs must be solved to accomplish this.

2.3 Boundary Conditions

The edges of the rod are crucial in determining how heat flows due to the boundary conditions. These conditions are required to calculate the temperatures at the boundaries, which is vital for solving the system of ODEs obtained from the spatial discretization.

- **Left-side Boundary** (Heating Source)
  At the left boundary n = 1, a heat source is applied. Instead of a simple derivative condition, we introduce a source term that adds heat to the system. The boundary condition at n = 1 is modified as follows,

  $$-\lambda \left.\frac{\partial T}{\partial x}\right|_{n=1} = u(t),$$

  The thermal conductivity of the material, denoted by λ, is a key factor in this equation. The expression $\frac{\partial T}{\partial x}|_{n=1} = u(t)$, signifies the temperature gradient at the left boundary. The term $u(t)$ represents the heat flux, which may vary depending on the specific heat source. This heat flux denotes the rate at which heat is introduced to the system at the left boundary.

  In the discrete system, this is approximated by a central difference at the left boundary,

  $$\frac{\partial^2 T_1}{\partial x^2} = \alpha \frac{2T_2 - 2T_1}{\Delta x^2} + \frac{2\alpha u}{\lambda \Delta x}.$$

  The term $\frac{2T_2 - 2T_1}{\Delta x^2}$ represents the second derivative using an imaginary point outside the boundary, $\frac{2\alpha u}{\lambda \Delta x}$ represents the heat added from the heat source.

- **Right-side Boundary** (Neumann Boundary Condition)
  At the right boundary n=N, a Neumann boundary condition is applied. A Neumann boundary condition specifies the derivative of the temperature at the boundary, which can represent a fixed heat flux. For example, if the boundary is insulated (no heat flow), the condition is,

$$\left.\frac{\partial T}{\partial x}\right|_{n=N} = q,$$

  For example, if q=0 (insulated boundary), this becomes

$$T_N = T_{N-1},$$

  This implies that the temperature at the last node is the same as that at the previous node, maintaining the insulation condition.

  The discrete system approximates this by a central difference at the right boundary.

$$\frac{\partial^2 T_N}{\partial x^2} = \frac{2T_{N-1} - 2T_N}{\Delta x^2}.$$

Now that we have established the boundary conditions, we have a complete set of equations that control the temperature at the interior and boundary points of the rod. The conversion of these equations into matrix form allows for the efficient numerical solution of the system.

2.4 Matrix Form of the Discretized System

The semi-discrete heat equation can be solved numerically by transforming the system of equations into matrix form, which can be written as the equation below. This enables us to calculate the temperature evolution at each spatial point effectively using numerical techniques. The matrix form makes condensing the equations obtained from the central difference approximation and the boundary conditions into a solvable system possible,

$$\frac{dT_n}{dt} = M \cdot T + F,$$

The vector T_n represents the temperatures at each spatial point, where $T_n = [T_1, T_2, \ldots, T_N]^T$, and N represents the number of spatial grid points. The matrix M corresponds to the finite difference approximation of the second derivative concerning space, derived using a central difference scheme. This scheme calculates the second temperature derivative by considering differences between adjacent points, resulting in a tridiagonal matrix representing heat conduction between neighboring nodes. In a one-dimensional domain, the matrix M contains entries that illustrate interactions between neighboring points, and its typical

structure includes coefficients that multiply temperature differences based on the spatial discretization step size Δx and the thermal diffusivity α,

$$M = \frac{\alpha}{\Delta x^2} \begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & -2 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & -2 \end{bmatrix} [10]. \tag{2.2}$$

Furthermore, vector F considers the impact of external heat sources, especially at the domain boundaries. In this instance, F illustrates a heat source applied to the left boundary, with the first entry being non-zero and the rest being zero. The following expression represents the heat source term,

$$F = \frac{\alpha}{\Delta x} \begin{bmatrix} \frac{2u}{\lambda} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Now that we have the heat equation expressed in matrix form, including boundary conditions and external heat sources(u), we can proceed to solve the system numerically. Before calculating the temperature changes over time, it's necessary to define the initial temperature distribution along the rod, given by the initial conditions.

2.5 Initial Conditions

The initial conditions at the beginning of the heating process define the temperature distribution in the rod. This initial temperature profile is used as the initial point for solving the system of ODEs obtained from the matrix representation of the heat equation. Regardless of whether the rod begins at a consistent ambient temperature or with a pre-heated profile, the initial conditions are crucial for precisely simulating the temperature changes,

$$T(x, 0) = T_0 = 300K,$$

for $x \in [0, L]$. After specifying the initial temperature distribution, the system can be completely described in its matrix form. The matrix system captures changes in temperature along the rod by considering heat diffusion and the impact of boundary conditions. The matrix form allows for an efficient and compact representation to solve the system numerically.

2.6 The final matrix system

The final matrix system is given below,

$$\frac{d}{dt}\begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_{N-1} \\ T_N \end{bmatrix} = \frac{\alpha}{\Delta x^2}\begin{bmatrix} -2 & 2 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ 0 & 0 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -2 \end{bmatrix}\begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_N \end{bmatrix} + \frac{\alpha}{\Delta x}\begin{bmatrix} \frac{2u}{\lambda} \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.3}$$

The heat diffusion through the internal points,
The heat source applied at the left boundary,
The Neumann boundary condition applied at the right boundary.

The heat diffusion and boundary conditions are included in the ultimate matrix system, but an effective numerical solver is required to calculate the temperature changes over time. Considering the potential stiffness of the heat equation, we decided on advanced solvers such as Rodas5, which are particularly skilled at effectively and accurately handling stiff systems.

2.7 Solvers

Rodas5 is particularly suited for stiff problems like the heat equation because it efficiently handles the stiffness often present in the time evolution of heat distribution [11]. A high-order method can achieve accurate results with more significant time steps than straightforward methods such as the Runge-Kutta method [12].

When we utilize the Rodas5 solver to solve the system, it precisely predicts the temperature changes, but actual measurements in the real world are rarely perfect. To account for the uncertainty present in real-world data, we add Gaussian noise to the system. This variability emulates the uncertainty in temperature readings, ensuring our model closely mimics real-life conditions.

2.8 Gaussian Noise

Introducing Gaussian noise into the system introduces uncertainty and variation to the data. Adding the noise term, ϵ, affects the temperature vector T, resulting in a new vector,

$$T_{noisy} = T_{original} + \epsilon.$$

This noise term, sampled from a Gaussian distribution ($\mathcal{N}(0, \sigma^2)$)with zero mean and variance σ^2 ($\sigma = 3$), plays a crucial role in capturing the uncertainty in measurements, thereby enhancing the precision of our modeling and simulations [13, 14].

In our computational model, we include a stochastic noise term ϵ, which accounts for measurement uncertainty. This ϵ is selected from a Gaussian distribution with zero mean and variance σ^2, represented as ($\epsilon \sim \mathcal{N}(0, \sigma^2)$), where $\sigma = 3$. The addition of this noise component accounts for the inherent variability in measured data, thereby improving the statistical fidelity and robustness of our simulations. Our model's accuracy and predictive capability are enhanced by explicitly incorporating the random variations measured in experimental results.

Incorporating this noise alters the system's heat equation of the heat conduction process. The system with noise is represented as,

$$\frac{d}{dt}T_{\text{noisy}} = M \cdot T_{\text{noisy}} + F + \epsilon,$$

here M denotes the matrix for the discretized spatial derivatives, F is the vector that considers external heat sources, and ϵ introduces stochastic variation to the system.

In the chapter, we laid the groundwork for creating a theoretical framework simulating heat conduction in a rod. We formulated a discretized model using matrices and incorporated external heat sources and Gaussian noise to mirror real-world scenarios. However, this theory provides a solid understanding of the system's behavior, the later application in chapter 5. This will involve simulations, visual representations, and pseudocode to facilitate a step-by-step understanding.

Controller Design and System Dynamics 3

This chapter discusses the development of a control mechanism that regulates the heating component located on the left side of the rod to maintain a specific temperature at the opposite end. We will initially outline the system in open-loop and closed-loop setups and then explain the functionality of the proportional controller within the closed-loop setup.

3.1 Open-loop System

The heating element on the left side of the rod is controlled independently without taking feedback from temperature sensors along the rod into account. The heat input is applied based on a predetermined value, and the system's behavior is not modified based on actual temperature measurements. An Open-loop system does not have automatic adjustment, which can lead to significant inaccuracies if external conditions, such as heat loss or variations in thermal properties, change [15]. This underscores the urge for a more advanced control mechanism.

3.1.1 Open-Loop Dynamic

In an Open-loop system, the heating element at the left end of the rod operates without any feedback from the temperature sensors. The heat input is administered based on a pre-established plan, which relies on system behavior estimations. The system does not adjust the heat input based on the measured temperature at the insulated end or in the middle of the rod.

The heat input, $q_in(t)$, is applied according to a fixed function of time,

$$q_in(t) = f(t).$$

The function of $f(t)$ can vary depending on the use. For instance, $f(t)$ might denote a consistent heat input over time, like $f(t) = C$, where C stays constant throughout the process. Alternatively, it could represent a function dependent on time, such as $f(t) = C + at$, with the heat input gradually increasing over time, mirroring a situation where the heating intensity ramps up as the process advances. In other instances, $f(t)$ might represent a more intricate, cyclical function, like $f(t) = Asin(\omega t)$, where the heat input oscillates periodically, simulating a scenario with controlled thermal cycling.

In all these scenarios, the system follows a predetermined heating variable. It lacks the capability to adapt to real-time temperature changes or external disturbances, consequently limiting its adaptability and precision.

3.1.2 System Behavior

The rod experiences changes in temperature distribution over time due to both external heat input and the thermal properties of the material, such as thermal diffusivity (α). If there is no feedback control, any disruptions, like external cooling or variations in material properties, are not countered. As a result, these factors may hinder the system from reaching the intended temperature at the insulated end of the rod, possibly causing failure to achieve the desired thermal state.

3.1.3 Limitations of Open-Loop Control

The main drawback of an Open-loop system is its inability to adapt to changes or uncertainties [16]. For example, if the system fails to reach the target temperature at the right end, there is no mechanism to adjust the heat input and correct the error.

While Open-loop systems have limitations, they are still extensively used in specific applications due to various benefits. Their simplicity in design and implementation, as they do not require sensors, feedback loops, or complex algorithms, makes them advantageous when system complexity needs to be minimized. Moreover, Open-loop systems are generally more cost-effective because they require fewer components and reduced maintenance.

In stable environments, Open-loop systems perform well as they can function reliably without feedback if external conditions are predictable and controlled. They also provide faster response times, ideal for applications where quick action is necessary, as they do not depend on feedback to adjust their input. Additionally, they are less prone to instability since there is no feedback loop, eliminating the risk of instability caused by poorly tuned feedback mechanisms.

Open-loop systems can be sufficient for applications where high precision is not essential. In scenarios where minor deviations in performance are acceptable, the added complexity of a feedback-based system may not be necessary. These factors make Open-loop systems a practical solution for many applications, particularly in stable, predictable environments. However, for critical situations where adaptability and precise control are essential, a Closed-loop system provides a more effective and reliable solution.

3.2 Closed Loop System with Proportional Control

The Closed-loop system incorporates a feedback mechanism for operation [17]. This feedback mechanism is implemented by continuously measuring the temperature with a sensor on the right side of the rod. Based on this measurement, the controller then adjusts the heat source on the left side to achieve the desired target temperature (setpoint) on the right side, denoted as r.

The closed-loop system benefits from using feedback because it enables more precise responses to environmental changes or disturbances, enhancing its adaptability and reliability compared to an open-loop system. This is achieved by implementing a Proportional Controller (denoted as P-Controller) to maintain precision in reaching the desired temperature target.

3.2.1 Closed-Loop Dynamic

The P-Controller, a crucial element of the closed-loop system, offers significant adaptability in managing the temperature profile of the rod. It optimizes sensor temperature measurements, particularly at critical points along the rod's insulated right end, and adjusts the heat input at the left end accordingly. Continuously assessing the current temperature against the desired temperature (r), the controller adjusts the heat input to reduce the difference between the two.

The primary goal of this Closed-loop setup is to ensure that the temperature at the end of the rod consistently reaches and maintains a specific desired target temperature, denoted as r.

3.2.2 P-Controller

The P-Controller is a primary type of feedback controller used in control systems to minimize the difference between a desired target point and the system's output [18]. Let us break down how the P-controller operates according to the principles detailed.

The system consistently measures and evaluates the current output against the intended setpoint. The discrepancy between the setpoint and the current output is known as the error,

$$e(t) = r - y(t).$$

The P-Controller modifies the control input $u(t)$ relative to the error. The following formula can be used to determine the control input, giving a detailed explanation of how the P-controller functions based on the principles explained earlier,

$$u(t) = K_p \cdot e(t) = K_p \cdot (r - y(t)). \tag{3.1}$$

The proportional gain represented as (K_p) adjusts the heat input based on the measured temperature of the target. When the measured temperature falls below the target, the control action increases the heat input, and when it surpasses the target, it reduces the heat input. The correction made is proportional to the size of the error, ensuring that the system continuously adapts to maintain the desired temperature [19].

P-Controllers find widespread use in industrial settings, such as HVAC systems, ovens, and manufacturing processes, where the maintenance of a stable temperature is essential for achieving peak performance. This straightforward but efficient control approach guarantees that the system can promptly adjust to deviations, gradually diminishing the error until the targeted temperature is reached.

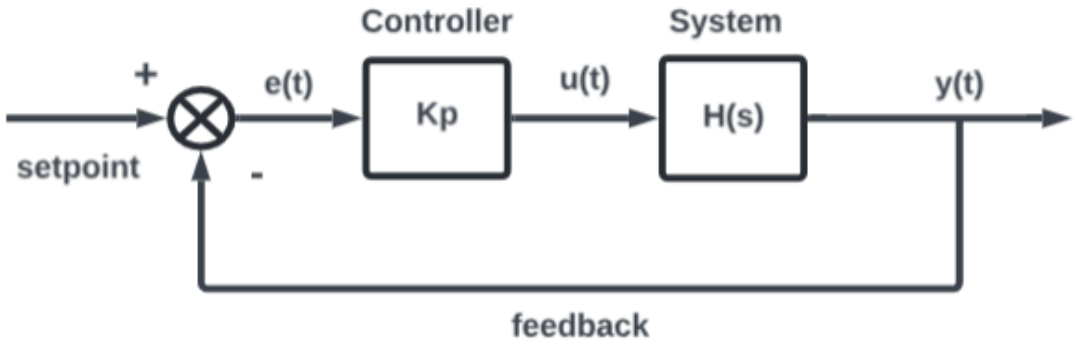

Figure 3.1: P-Controller

The system receives the control signal $u(t)$ and adjusts its behavior, such as heating the rod. The modified output is then looped back into the system to recalculate the error. The system uses a feedback loop to constantly adjust the input to reduce errors over time.

3.2.3 Behaviour of a P-Controller

The value of the controller gain K_p has a significant impact on the implementation of a P-Controller. A higher K_p results in a more assertive reaction to errors, leading to swift corrections to steer the system toward the desired temperature. This assertiveness can result in overshooting and oscillations.

Conversely, a low K_p results in a more cautious response, where the P-Controller makes slower adjustments to prevent overshooting. However, this caution may lead to slower convergence to the target temperature, highlighting the need for a balanced approach.

Finding the right balance for K_p is crucial for optimal system performance. The response of the system to disturbances can be significantly impacted, as well as its effectiveness in maintaining the desired temperature.

Selecting the correct value for K_p is crucial in maintaining a balance between responsiveness and stability, which is necessary for the system to reach the target temperature without excessive oscillations or delays.

The P-Controller is crucial for regulating the system's temperature, and accurately determining the parameters K_p and λ is vital for maximizing performance. In the upcoming chapter, we will investigate how inverse techniques are employed to establish these critical parameters using temperature data from the system .

Parameter Identification Using Inverse Methods

4

Inverse techniques are used when there is output data, such as temperature measurements, and the objective is to determine the underlying parameters, like K_p and λ, that impact the system's behavior. These methods are commonly used in systems where direct measurement of the parameters is not feasible or practical.

Inverse methods are crucial in identifying unknown parameters in a system when only the output data, such as temperature measurements, are accessible. In our scenario, parameters such as the proportional gain K_p and thermal conductivity λ, which impact the performance of the P-Controller and the heat equation, must be precisely determined to ensure optimal system behavior. These techniques enable us to reverse engineer the measured data to reveal the values that best describe the underlying dynamics of the system.

The parameters K_p, which controls the responsiveness of the P-Controller, and λ, the thermal conductivity that determines heat conduction in the rod, are crucial for accurate control and temperature regulation. Discovering these parameters through inverse methods ensures the system can maintain desired thermal conditions under various environmental factors.

This chapter details how to use inverse methods for parameter identification, offering a straightforward approach from collecting temperature data to estimating parameters. Implementing these methods will enhance the control system's performance, ensuring stability and responsiveness in practical situations.

4.1 Inverse Problem Formulation

Our setup has a rod featuring a heating unit and multiple temperature sensors positioned at various locations. The objective is to utilize the temperature data obtained from these sensors to estimate the proportional gain K_p of the controller and the thermal conductivity λ of the rod material.

This will be accomplished through inverse techniques, which entail solving the inverse problem of adjusting the unknown parameters to fit the measured data to a model.

4.1.1 Forward and Inverse Problem

When dealing with heat conduction problems, we can consider the system from two approaches: the forward problem and the inverse problem [20]. The forward problem entails using known system parameters to predict the system's behavior. On the other hand, the inverse problem involves using monitored system behavior to derive unknown parameters.

- **Forward Problem:** The forward problem aims to predict the system's result, particularly the temperature profile along the rod, by relying on identified system parameters such as K_p(the gain of the P-Controller) and λ(thermal conductivity).
  The setup entails simulating the physical system using established equations like the heat conduction equation in this situation. We use these known parameters to predict the temperature evolution over time given the input (e.g., applied heat). The expected outcome is the predicted temperature distribution T(x,t) along the rod, based on the input data and the established model parameters.

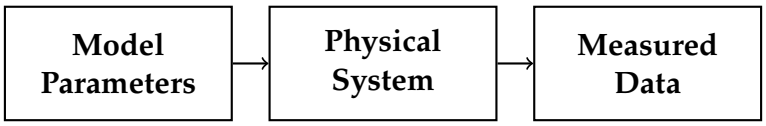

Figure 4.1: Forward Problem

  Reproducing the physical system through simulation is possible using well-defined system parameters like K_p and λ. Through the mathematical formulation of the system, for example, by utilizing established equations such as the heat equation to model the rod, it becomes feasible to compute the temperature progression based on the input provided (heat). The result of this simulation is the predicted temperature distribution $T(x, t)$.

- **Inverse Problem:** In the inverse problem, we lack knowledge of the system parameters and seek to calculate these parameters based on the measured system data. Our configuration involves gathering temperature readings from sensors positioned throughout the rod. The goal is to ascertain the most suitable values for K_p and λ that can account for the measured temperature distribution.

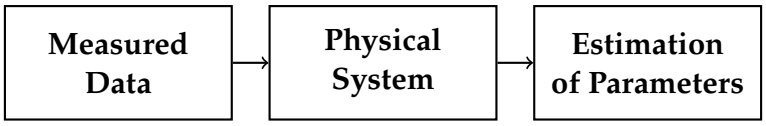

Figure 4.2: Inverse Problem

  The process starts with gathering the real-time data obtained from the system, such as the temperature readings at various locations on the rod as measured by the sensors. Using the system's model, for instance, the heat equation, efforts are made to identify the most suitable parameters that would effectively account for the collected data. The model is iteratively run with parameter adjustments to minimize the variance between the projected temperatures and the actual measurements.

The solution to this problem necessitates a method for assessing how accurately the model's projected temperatures correspond to the actual measurements taken from the system. This is where the loss function becomes significant.

4.1.2 Loss Function

In order to solve the inverse problem and determine the unknown parameters K_p and λ, it is crucial to assess the degree to which the simulated results match the actual temperature readings from the system. This comparison is measured using a loss function, which indicates the difference between the predicted temperature profiles and the measured data. We can assess the degree to which the parameter estimate matches the actual system.

A loss function usually offers a mathematical structure for measuring the difference between the predicted outcomes obtained from solving the forward problem and the actual data gathered from temperature sensors. Minimizing this variance enhances the accuracy of parameter estimates. Commonly used loss functions include the Mean Squared Error (MSE) and Mean Absolute Error (MAE) [21].

Mean Squared Error (MSE)
It is a widely used statistical measure for assessing model accuracy. This is achieved by computing the average of the squared variances between the actual and predicted values by the model. When dealing with inverse problems and predictive modeling, MSE is valuable for evaluating the alignment between the model predictions and real-world data. The MSE formula is

$$J(K,\lambda) = \frac{1}{N}\sum_{i=1}^{n}(T_{\text{measured}}(x_i,t) - T_{\text{model}}(x_i,t;K,\lambda))^2 \text{ [22]}.$$

The total number of measurements is represented by N in this equation. $T_{\text{measured}}(x_i,t)$ denotes the temperature measured by sensors, while $T_{\text{model}}(x_i,t;K,\lambda)$ represents the temperature predicted by the model. The parameters K and λ pertain to the proportional gain and other model parameters, respectively. The objective of minimizing MSE is to decrease the total squared error between the model's predictions and the actual data.

The shape of the MSE loss function obtained from the simulation setup is illustrated in Figure 4.3, which demonstrates how MSE behaves as the model parameters are adjusted. It also shows the behavior of the error surface when testing different parameter values.

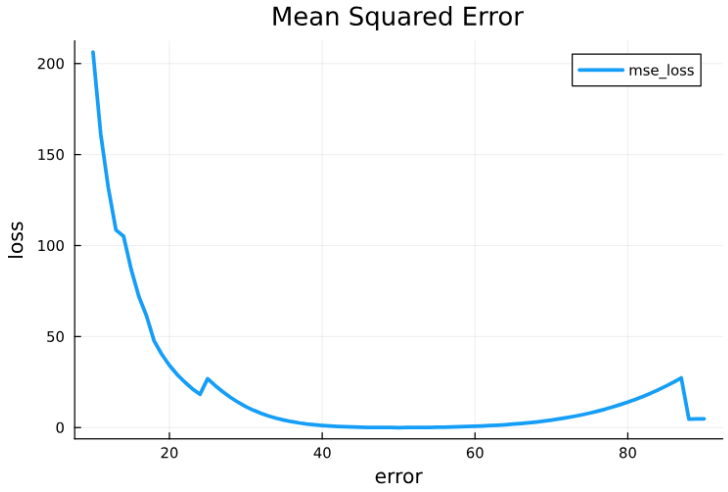

Figure 4.3: Loss function of MSE

Mean Absolute Error (MAE)
Model performance evaluation involves using another measure known as **Mean Absolute Error (MAE)** [23]. In contrast to MSE, which involves squaring the disparities between predicted and actual values, MAE determines the mean of the absolute disparities, making it less affected by significant substantial errors. Regardless of their magnitude, MAE treats all errors equally. The formula for MAE is

$$J(K, \lambda) = \frac{1}{N} \sum_{i=1}^{n} |T_{\text{measured}}(x_i, t) - T_{\text{model}}(x_i, t; K, \lambda)| \text{ [24].}$$

Again, $T_{measured}(x_i, t)$ denotes the temperature measured by sensors, while $T_{model}(x_i, t; K, lambda)$ represents the temperature predicted by the model. The objective of MAE is to offer a more even evaluation of prediction inaccuracies while avoiding amplifying larger differences.

Figure 4.4 below illustrates the relationship between the MAE loss function and the error values produced by your simulation setup. It is evident that the actual value of $\lambda = 50$ is located at the lowest point of the U-shaped curve, confirming that the utilization of MAE effectively attains the desired balanced error distribution for your system. Therefore, it is optimal for minimizing the absolute error between the predicted and measured values.

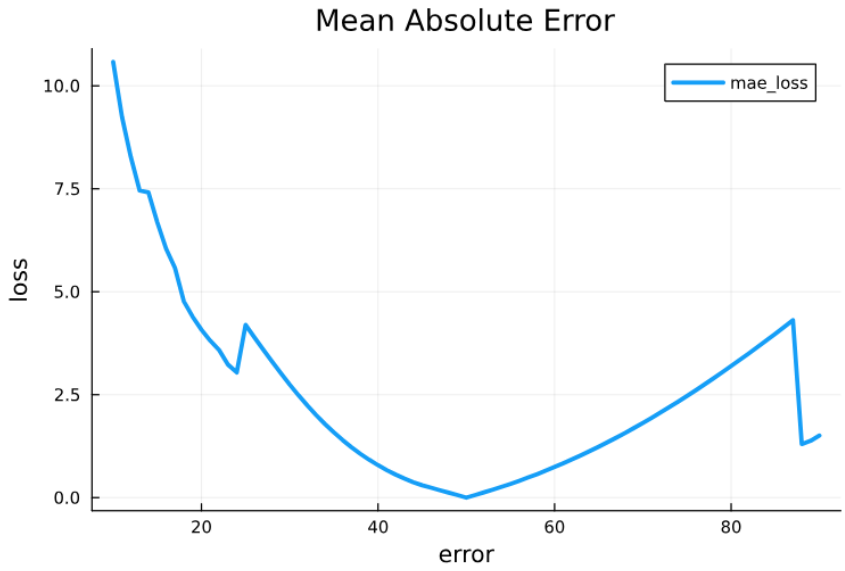

Figure 4.4: Loss function of MAE

The use of MAE involves absolute variances, but the resulting loss function may behave similarly to a quadratic function near the minimum, particularly as it approaches the true parameter value.

4.2 Optimization Methods

After establishing the loss function, such as the Mean Absolute Error (MAE), our next objective is to minimize this function to determine the optimal estimates for the unknown parameters $\lambda$and K_p. The aim is to identify the parameter values that ensure the model-predicted temperature profile closely matches the measured data, thereby addressing the inverse problem.

To accomplish this minimization, we utilize optimization methods, which are also referred to as optimizers [25]. This method iteratively modifies the parameters $\lambda$and K_p, seeking the combination of values that minimize the MAE and reduce the variation between the model's predictions and the actual measurements.

4.2.1 Nelder Mead Method

The Nelder-Mead algorithm is precious when calculating derivatives is challenging, as is often encountered in real-world problems. It falls into the category of "derivative-free" techniques. It utilizes a simplex, a geometric form such as a triangle or tetrahedron, to navigate the parameter space and locate the best solution [26].

How it works? Unlike many other methods, the algorithm refrains from directly ascending or descending by following the steepest gradient. Instead, it alters the simplex by expanding, contracting, or reflecting its points to reach a configuration that minimizes the loss function.

Mathematical Formulation:

1. **Simplex Vertices:** Let $p_1, p_2, ..., p_{n+1}$ be the vertices of the simplex, where each $p_i = [K_i, \alpha_i]$ is a vector of λ and K_p parameters .

2. **Objective Function Ordering:** Compute and order the objective function $J(p_i)$ at each vertex:
$$J(\mathbf{p}_1) \leq J(\mathbf{p}_2) \leq \cdots \leq J(\mathbf{p}_{n+1})$$
In this case, p_1 represents the optimal point with the lowest loss, while p_{n+1} represents the point with the highest loss.
3. **Centroid Calculation:** Taking the information provided, we can find the centroid c of the points by excluding the lowest vertex p_{n+1}. The centroid represents the "average" of the top points and can be computed in the following manner
$$\mathbf{c} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{p}_i.$$

4. **Reflection:** Reflect the worst vertex through the centroid

$$\mathbf{p}_r = \mathbf{c} + \alpha_r(\mathbf{c} - \mathbf{p}_{n+1}),$$

where $\alpha_r > 0$ is the reflection coefficient.

Expansion/Contraction/Reduction: Depending on the value of $J(P_r)$, decide whether to expand, contract, or reduce the simplex to reach the minimum.

Advantages: It is helpful for optimization problems where the loss function is not smooth or where derivatives are unavailable [27].

Application in Parameter Identification: Nelder-Mead can adjust K_p and λ iteratively, comparing the predicted and measured temperatures and modifying the simplex until the error is minimized.

4.2.2 Conjugate Gradient Method

The method of Conjugate Gradient is utilized to optimize functions by minimizing them, especially when dealing with a large number of parameters. It represents an enhancement over the conventional gradient descent approach by incorporating insights from previous iterations to adapt the search direction for the minimum, thereby increasing its effectiveness[28].

Mathematical Formulation:

1. **Initialization:** Start with an initial guess $\mathbf{p}_0 = \left[K_0, \lambda_0\right]$ and compute the initial gradient $\mathbf{g}_0$:

$$\mathbf{g}_0 = \nabla J(\mathbf{p}_0)$$

Set the initial search direction as $\mathbf{d}_0 = -\mathbf{g}_0$.(negative of the gradient)

2. **Parameter Update:** At each iteration k, update the parameter vector:

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \lambda_k \mathbf{d}_k$$

where α_k is the step size found by a line search that minimizes $J(\mathbf{p}_k + \lambda \mathbf{d}_k)$.

3. **Gradient Update:** Compute the new gradient $\mathbf{g}_{k+1}$ at $\mathbf{p}_{k+1}$:

$$\mathbf{g}_{k+1} = \nabla J(\mathbf{p}_{k+1})$$

4. **Conjugate Direction:** Update the search direction using the conjugate gradient formula:

$$\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k$$

where β_k is given by:

$$\beta_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k}$$

5. **Iteration:** Continue iterating until the norm of the gradient $\|\mathbf{g}_{k+1}\|$ is sufficiently small, indicating that the minimum has been reached.

Advantages: The Conjugate Gradient Method is advantageous because it has the potential to converge more rapidly than regular gradient descent, particularly in scenarios with numerous parameters. As a result, it is the preferred approach for tackling extensive problems, such as optimizing parameters in a system with multiple variables [29].

Application in Parameter Identification: When it comes to estimating parameters such as k_p and λ, the Conjugate Gradient Method has the ability to iteratively modify these values in order to minimize the discrepancy between predicted and measured temperatures. This method improves the optimization procedure, making it easier to reach the right parameters more quickly than traditional approaches.

4.2.3 Newton's Method

Newton's Method, being a second-order optimization technique, utilizes both the gradient (first derivative) of the loss function and the Hessian matrix (second derivative) [30]. This characteristic endows it with the capability to rapidly find the minimum of a function, particularly when the function exhibits smooth and well-behaved behavior near the minimum.

Mathematical Formulation:

1. **Gradient and Hessian Calculation:** At each iteration k, compute the gradient $\mathbf{g}_k$ and the Hessian matrix $\mathbf{H}_k$:

$$\mathbf{g}_k = \nabla J(\mathbf{p}_k), \quad \mathbf{H}_k = \nabla^2 J(\mathbf{p}_k)$$

2. **Newton Step:** Update the parameter vector using the Newton step:

$$\mathbf{p}_{k+1} = \mathbf{p}_k - \mathbf{H}_k^{-1}\mathbf{g}_k$$

   Here, $\mathbf{H}_k^{-1}$ is the inverse of the Hessian matrix.
3. **Iteration:** Continue iterating until the parameter updates $\|\mathbf{p}_{k+1} - \mathbf{p}_k\|$ become sufficiently small, indicating that convergence has been achieved.

Matrix Representation:

- The update step in matrix form is:

$$\mathbf{p}_{k+1} = \mathbf{p}_k - \mathbf{H}_k^{-1}\mathbf{g}_k.$$

- The Hessian matrix $\mathbf{H}_k$ for two parameters (e.g., K and λ) is,

$$\mathbf{H}_k = \begin{bmatrix} \frac{\partial^2 J}{\partial K^2} & \frac{\partial^2 J}{\partial K \partial \lambda} \\ \frac{\partial^2 J}{\partial \lambda \partial K} & \frac{\partial^2 J}{\partial \lambda^2} \end{bmatrix} \text{[31]}.$$

Advantages: Newton's method demonstrates rapid convergence compared to gradient-based methods, particularly near the minimum, owing to its utilization of second-order information (the Hessian) for making more precise steps.

Application in Parameter Identification: In cases where the loss function, like MAE or MSE, shows smooth behavior, Newton's method can be highly effective for parameter identification, such as finding K_p and λ. The method utilization of gradient and curvature information enables it to rapidly fine-tune parameters to minimize the disparity between predicted and actual measurements.

Suppose the Hessian can be efficiently computed or approximated. In that case, Newton's method can expedite the refinement of estimates for K_p and λ compared to methodologies that solely rely on the gradient.

4.3 Automatic differentiation

Automatic Differentiation (AD) is a method that reliably calculates the derivatives of functions with precision and efficiency. This technique instills confidence in computing loss function gradients concerning parameters in optimization, machine learning, and control systems[32].

4.3.1 Concept of Automatic Differentiation

Automatic Differentiation involves decomposing a complicated function into basic operations like addition and multiplication and then using the calculus chain rule to calculate derivatives. Unlike symbolic differentiation, which offers a symbolic representation of the derivative, or numerical differentiation, which estimates the derivative using finite differences, AD computes precise numerical derivatives at a specific point. AD consists of two main modes:

- **Forward Mode:** Suited for cases where derivatives are needed for only a few inputs. Derivatives are forwarded from the inputs to the outputs in forward mode.
- **Reverse Mode:** Efficient for situations with a small number of outputs, such as a single loss function, and a large number of inputs, which are parameters. The reverse mode is frequently utilized in machine learning to propagate derivatives from the outputs to the inputs, such as gradients in backpropagation.

4.3.2 Chain Rule

The chain rule is an important aspect of automatic differentiation (AD) because it enables the propagation of derivatives through a series of intermediate operations. In the case of a

composite function $y = f(g(h(x)))$, the chain rule specifies the following,

$$\frac{dy}{dx} = \frac{dy}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{dx}.$$

When using forward mode AD, the chain rule calculates function values and their derivatives as they move from the input to the output.

4.3.3 Forward Mode Automatic Differentiation

In the forward mode of automatic differentiation (AD), derivatives are calculated simultaneously with evaluating functions. The computation is divided into intermediate variables when dealing with a general function $y = f(x_1, x_2, \ldots, x_n)$.

1. Define intermediate variables,

$$w_1 = h_1(x), w_2 = h_2(w_1), \ldots, w_m = h_m(w_{m-1}).$$

2. The derivative is propagated using the chain rule,

$$\frac{\partial y}{\partial x} = \frac{\partial w_m}{\partial w_{m-1}} \cdot \frac{\partial w_{m-1}}{\partial w_{m-2}} \cdot \cdots \cdot \frac{\partial w_1}{\partial x}.$$

The process in forward mode involves propagating derivatives from independent variables through intermediate operations, with calculations made at each step.

4.3.4 Example: Forward Mode AD

Let's consider the function $y = x_1x_2 + \sin(x_1)$ We want to compute the function's value and its derivatives concerning $x_1$and x_2.

Break the function into intermediate variables.
Breaking the computation into smaller steps involves using intermediate variables to compute derivatives. In our example, we can express the function by using intermediate variables,

$$w_1 = x_1,$$

$$w_2 = x_2,$$

$$w_3 = w_1w_2 = x_1x_2,$$

$$w_4 = \sin(w_1) = \sin(x_1),$$

$$w_5 = w_3 + w_4 = x_1x_2 + \sin(x_1)$$

Here, $w_5 = y$ is the final output of the function, which is y.

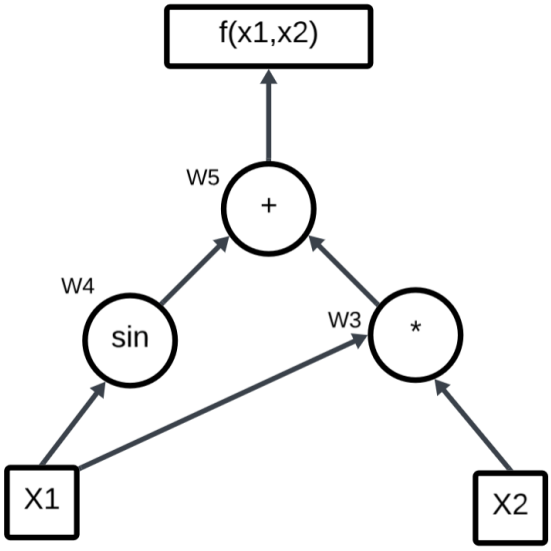

Figure 4.5: Forward Mode Automatic Differentiation

We have examined the application of Forward Mode Automatic Differentiation (AD) for calculating the function's value and its derivatives concerning the input variables. Efficiently computing derivatives is a crucial aspect of the optimization process because many optimization techniques, such as Conjugate Gradient and Newton's Method, depend on gradient information to facilitate the search for optimal parameters. Forward mode AD optimizes this process by automating derivative computation, enabling more precise and efficient parameter identification.

After establishing the groundwork in theory and providing the necessary tools to tackle the inverse problem and refine the loss function, our focus now shifts to applying these principles practically in heat conduction. In Chapter 5, we will put these optimization methods into action and evaluate their efficiency in identifying the unknown parameters K_p and λ. Furthermore, we will explore incorporating automatic differentiation (AD) into the optimization procedure to enhance the calculations' accuracy. Through the outcomes of this implementation, including visual representations and numerical comparisons, we aim to illustrate the system's response to various optimization methods and restrictions.

5 Implementation and Results

In this chapter, the results of the parameter identification approach for heat conduction in the rod system are detailed. Various optimization methods, including the Nelder-Mead, Conjugate Gradient, and Newton's method, are used to determine thermal conductivity (λ) and proportional controller gain (K_p). The parameters are evaluated based on precision, stability, and computational efficiency.

5.1 Open-Loop Results

This part will evaluate how well an open-loop heat conduction system performs using forward problem formulation. This setup delivers heat without utilizing feedback control to test the system's capability to maintain a consistent temperature and recognize its constraints. The main emphasis is on applying a constant heat source for the initial 10 seconds and then monitoring the thermal behavior after deactivating the heat source for the remaining duration of the simulation.

The simulation models a rod that is 0.2 meters long, with a thermal conductivity (λ) of 50 $[\frac{W}{m \cdot K}]$. The density (ρ) is 10,000 kg/m^3 and its specific heat capacity (c) is 1 $[\frac{J}{Kg \cdot K}]$.The spatial discretization (Δx) is set to 0.01 meters, dividing the rod into 21 spatial points. Initially, the temperature across the entire rod is uniformly distributed at 300 K. For the first 10 seconds; the heat input is applied constantly at a rate of1×10^4 W. Afterward, the heat input is turned off, and the system transitions into a cooling phase for the remainder of the 20-second simulation.

Solving the heat equation will involve using the Rodas5 solver, which is a stiff ODE solver, and this process should be completed within 20 seconds. The heat conduction equation is discretized using the finite difference method, with 21 spatial points along the rod, and the resulting ODE system is solved using Rodas5.

As shown in Figure 5.1, the temperature dynamics of the open-loop system demonstrate a transparent gradient during the heating phase, followed by uniform cooling after the heat input is removed. The left side of the rod is subject to a constant heat source during the initial 10 seconds of the simulation. As expected, the area nearest to the heat source experiences a rapid temperature rise while heat conduction gradually spreads toward the right side of the rod. This results in a temperature gradient along the rod's length, with positions farther from the heat source heating up more slowly. Figure 5.1 shows a swift temperature rise near the heat source during the first 10 seconds, followed by slower heat propagation toward the rod's right end. The color variation signifies the temperature changes over time, showcasing the effective temperature regulation and stability maintenance provided in the open-loop system.

After 10 seconds, the system enters the cooling phase once the heat input is turned off. The temperatures at points nearest the heat source decrease rapidly, while points farther away cool slower. This natural heat dissipation returns the rod to a temperature near its initial state by the end of the 20-second simulation.

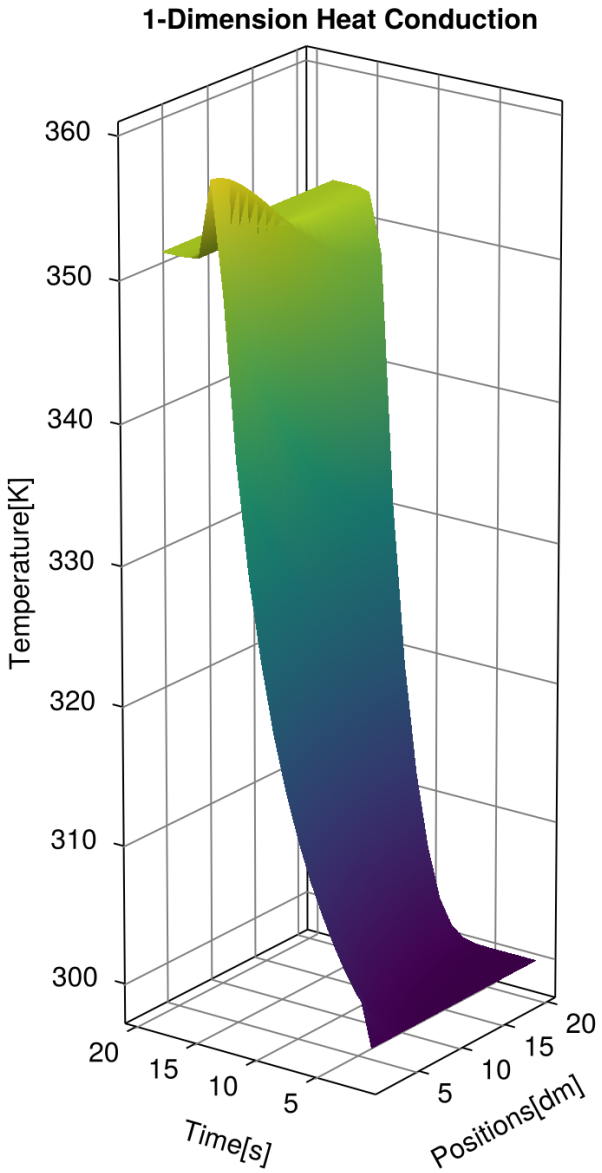

Figure 5.1: Temperature Distribution Over Time in the Open-Loop Heat Conduction System

The primary setback of the open-loop system is its inability to adjust to changing conditions. Once the heat input stops, the system cannot sustain high temperatures, making it ineffective for applications that require precise thermal regulation. In real-world applications—such as industrial processes or climate control maintaining stability and consistency in temperature control is crucial; this limitation underscores the need for advanced feedback mechanisms.

The starting point for the inverse problem is the temperature data obtained from the open-loop system. In the inverse problem, the main focus is estimating the unknown

system parameters, particularly the thermal conductivity λ, which cannot be measured directly. To address this, we compare the actual measured temperatures from the open-loop system to the predicted temperatures from a model generated using the forward problem. The Mean Absolute Error (MAE) loss function assesses the difference between these two data sets.

In the 4 Chapter, we analyzed how the Mean Absolute Error (MAE) loss is mathematically represented. In the last chapter, we investigated the application of the Mean Absolute Error (MAE) loss function and the Nelder-Mead optimization technique as mathematical instruments for solving inverse problems.

The Code 1 demonstrates the MAE loss function in Julia programming. It calculates the discrepancy between the predicted temperatures and the actual temperatures. The variable **data_ml** contains the predicted temperatures, whereas **data_orig** contains the original data obtained from the simulation. The MAE is determined by averaging the absolute variances between these two data sets. As stated in Equation 4.2, the MAE calculates the mean difference between the forecasted and real temperatures, which assists the optimization algorithm in modifying λ to reduce this discrepancy.

The following code initially defines the MAE loss function, which assesses the error for a specified thermal conductivity (λ). Subsequently, the optimize function utilizes the Nelder-Mead method to progressively refine the initial guess for λ (10.0), adapting it to minimize the MAE and determine the most precise value of λ. The conclusive outcome is preserved in **optimal**_λ, indicating the thermal conductivity most accurately corresponds to the measured data.

```
# Define the mean absolute error (MAE) loss function
   function mae_loss(λ)
       p_ml = λ
       sol = solve(prob, alg, p=p_ml, saveat=tsave)
       data_ml = Array(sol[1:end,:])
       err = (abs.(data_orig - data_ml))
       mae_loss = mean(err)
   return mae_loss

```

Code 1: Julia's implementation of the MAE Loss Function.

Nelder-Mead Method

The Nelder-Mead optimization method is applied to iteratively adjust the value of λ in order to minimize the MAE. This algorithm efficiently searches for the optimal λ using the measured temperature data as a reference. The goal of running the optimization process is to determine the value of λ that leads to the most minor error between the model predictions and the real-world data.

```
# Initial guess for the parameter λ (starting value for optimization)
    initial_λ = [10.0]
# Perform the optimization using the Nelder-Mead method to minimize the MAE
    result = optimize(mae_loss, initial_λ, NelderMead())
# Extract the optimal value of λ that minimizes the MAE
    optimal_λ = Optim.minimizer(result)

```

Code 2: Nelder-Mead Optimization Method to estimate the thermal conductivity λ.

Throughout the optimization process, the convergence of the MAE can be visually observed in the plot presented in Figure 5.2. The plot clearly illustrates the reduction in error over time as the algorithm approaches the true value of λ.

In conclusion of the optimization process, the calculated value of λ converges to 50, which aligns with the actual value employed in the system, confirming the precision of our inverse technique.

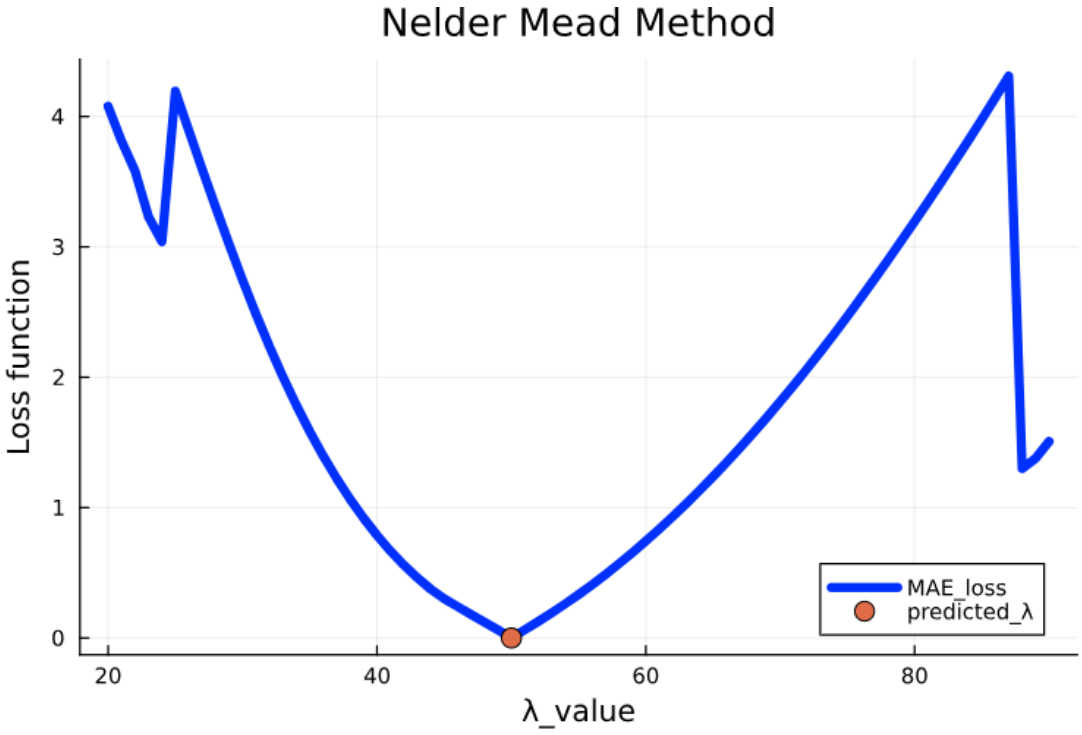

Figure 5.2: Nelder-Mead Method with open loop

The information above indicates that the Nelder-Mead method has provided an estimated value for λ close to the true value of 50. This confirms the precision of the inverse method, as the model's predictions closely align with the temperature measurements from the open-loop system. Through the successful estimation of λ, we have showcased the effectiveness of the inverse method in determining unknown system parameters using measured data.

Conjugate Gradient Method

The Conjugate Gradient Method is frequently utilized as an optimization algorithm, particularly for handling large-scale problems and smooth loss functions. Its application is aimed at reducing the Mean Absolute Error (MAE) between predicted temperatures and real data.

Code 3 represents the implementation of the Conjugate Gradient Method. The optimization process is guided by computing the gradient of the loss function, which is accomplished using ForwardDiff, a Julia package that automatically calculates derivatives. The gradient of the MAE loss function is computed and then supplied to the optimized function, along with an initial guess for λ (set to 25.0). The optimize function then iteratively adjusts λ using the Conjugate Gradient algorithm, searching for the value that minimizes the MAE.

```
# Compute the gradient of the MAE loss function with respect to λ
# ForwardDiff.gradient() automatically computes the gradient, helping the optimizer
    function gradient_mae_loss!(g, λ)
        g .= ForwardDiff.gradient(mae_loss, λ)
    end

# Initial guess for the parameter \(\lambda\), chosen as 25.0 based on preliminary assumptions
    initial_λ =  [25.0]
# Perform the Conjugate Gradient Method with a line search to ensure efficient progress toward minimizing the loss
# Use BackTracking Line Search to ensure effective step sizes
    result = optimize(mae_loss, gradient_mae_loss!, initial_λ, Optim.ConjugateGradient(linesearch=LineSearches.BackTracking()))
# optimal_λ  contains the best estimate for λ, minimizing the MAE
    optimal_λ = Optim.minimizer(result)

```

Code 3: Gradient-Based Optimization Using Conjugate Gradient Method for Thermal Conductivity Estimation.

The initial estimate for λ is established at 25.0, and the gradient of the MAE loss function is utilized in each iteration to guide the most suitable λ. The BackTracking Line search is a technique that ensures each iteration progresses toward minimizing the loss, preventing large or inefficient steps during optimization. After the optimization process is completed, the **optimal**_λ variable holds the finest approximation for λ.

The results of the application of the Conjugate Gradient Method to estimate λ are presented in Figure 5.3. The Figure illustrates the relationship between the loss function and different values of λ, with red circles along the curve representing the predicted values of λ as the method progresses. The method effectively converges towards the true value of λ, as reflected by the decreasing loss.

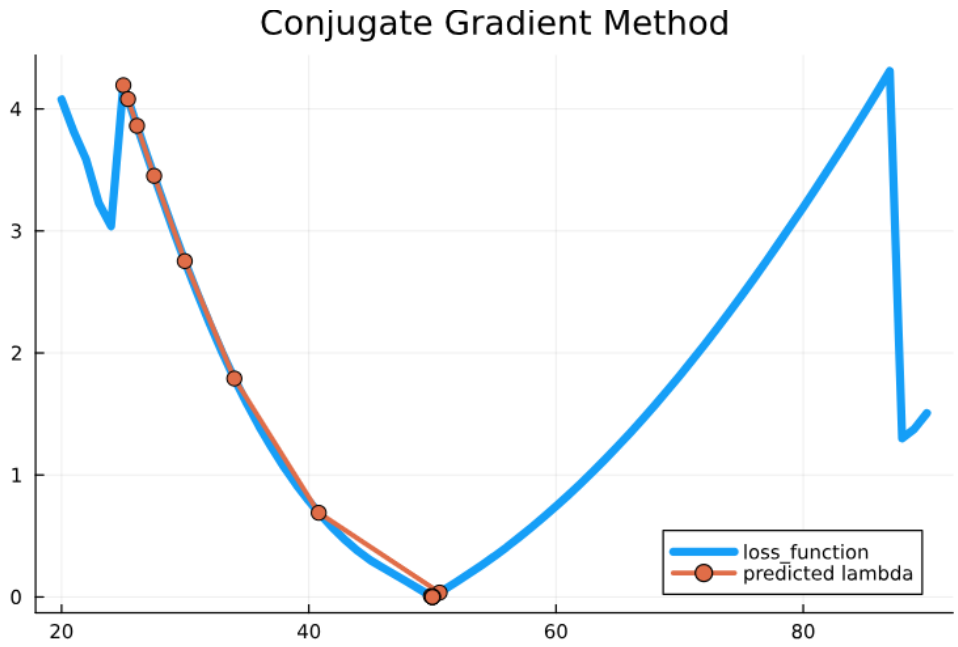

Figure 5.3: Conjugate Gradient Method with open loop

As observed in the plot, the Conjugate Gradient Method identifies the λ that minimizes the MAE, with the predicted values approaching the true value of approximately 50. This showcases the algorithm's ability to minimize the error between predicted and measured temperatures, resulting in an accurate estimate of the thermal conductivity.

Newton's Method

Newton's Method is a robust optimization technique that leverages both the gradient and the Hessian matrix of the loss function to determine the optimal parameter value. Unlike the Conjugate Gradient Method, which relies solely on the gradient, Newton's method takes a step further by incorporating second-order derivative information (Hessian) to guide the optimization process. This additional information often leads to quicker convergence, especially for smooth and well-behaved problems.

The code provided below illustrates the application of Newton's Method in Julia programming. The **hessian_mae_loss!** function calculates the Hessian matrix of the MAE loss function using the ForwardDiff package. This matrix depicts the second-order derivatives of the loss function and offers valuable curvature information, **aiding the algorithm in making more informed updates to λ.**

The initial assumption for λ in the code is 10.0. Newton's Method is then applied using the optimize function, which incorporates both the gradient and Hessian matrix of the loss function. The BackTracking line search is used to find the correct step size for every iteration. The optimal value of λ is saved in **optimal**_λ, indicating the most accurate estimation for the thermal conductivity.

```julia
# Define a function to compute the Hessian (second-order derivative) of the MAE loss function
# ForwardDiff.hessian() automatically computes the Hessian using the input λ and the mae_loss function.
    function hessian_mae_loss!(h, λ)
        h .= ForwardDiff.hessian(mae_loss, λ)
    end
# Set an initial guess for the parameter λ. This starting point (10.0 in this case) is required by the optimization algorithm.
    initial_λ =  [10.0]
# Perform optimization using Newton's Method.
    result = optimize(mae_loss, gradient_mae_loss!, hessian_mae_loss!, initial_λ, Newton(linesearch = BackTracking())
# optimal_λ contains the best estimate for λ, minimizing the MAE
    optimal_λ = Optim.minimizer(result)

```

Code 4: Newton's Method with Gradient and Hessian Calculation for Thermal Conductivity Estimation.

The results of applying Newton's Method can be visualized in Figure 5.4. The Figure shows how the loss function (MAE) is plotted against different values of λ, with red circles representing the estimated λ values at each iteration of the optimization process. As the algorithm advances, the loss function decreases, and the estimated λ approaches the true value.

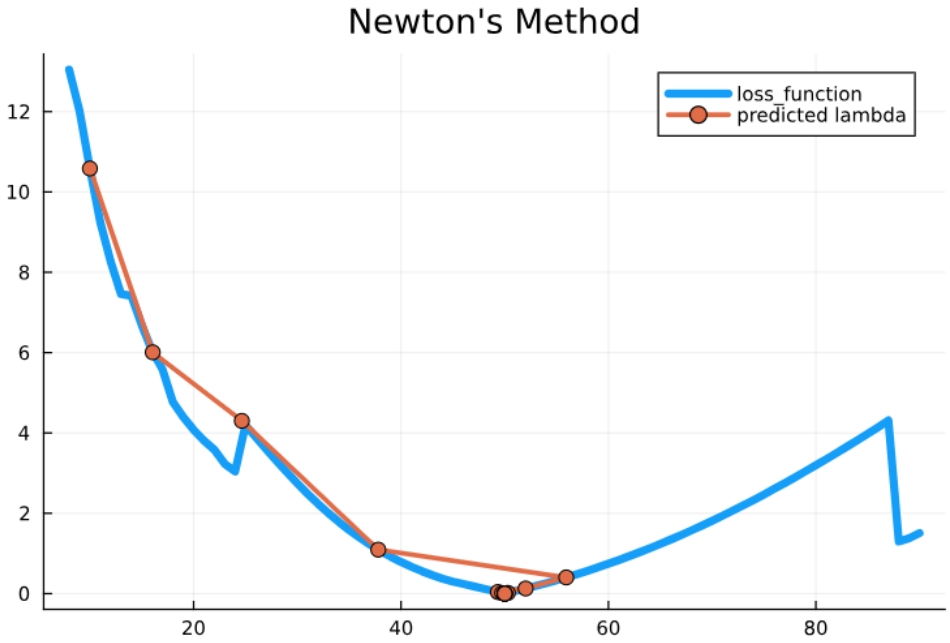

Figure 5.4: Newton's Method with open loop

The method effectively minimizes the loss function, highlighting the advantage of integrating both the gradient and Hessian in the optimization process. In comparison to gradient-based methods such as Conjugate Gradient, Newton's Method often converges more rapidly due to the additional curvature information provided by the Hessian matrix. In this instance, the method accurately estimates λ with fewer iterations, as evidenced by the significant reduction in the loss function.

Following the analysis of the open-loop system and the utilization of various optimization techniques to estimate the thermal conductivity λ, we now shift our focus to the closed-loop system, which integrates feedback control to achieve more accurate temperature regulation. Within the closed-loop configuration, a P-Controller is utilized to dynamically modify the heat input based on real-time temperature readings. In contrast to the open-loop system, where the heat input remains constant, the closed-loop system continuously adapts to maintain the desired temperature, rendering it more flexible and suitable for real-world scenarios where precision and stability are crucial.

The upcoming sections will explore the implementation of the P-Controller and showcase its efficacy in regulating the system's temperature.

5.2 Closed-Loop Results

The temperature of the rod in the closed-loop system is dynamically regulated using a P-Controller to ensure it reaches a target temperature of 500 K at the right end. In contrast to the open-loop system where the heat input was fixed and uncontrolled, the closed-loop system adjusts the heat input in real-time based on temperature measurements, making it adaptive and more responsive.

The P-controller functions by continually comparing the current temperature on the right side of the rod to the reference target temperature of 500 K. If the measured temperature is lower than this target, the controller increases the heat input at the left end of the rod to raise the temperature. Conversely, if the temperature exceeds the target, the controller decreases the heat input to stabilize the system. The system uses this feedback loop to keep the temperature constant by adjusting to changes in a dynamic way.

In Figure 5.5, the temperature distribution along the length of the rod over time is illustrated under the closed-loop system. The plot shows how the system rapidly approaches the target temperature of 500 K, with heat being concentrated near the left side and gradually dissipating along the rod as time progresses. The variation in color signifies the temperature changes over time, showcasing the effective temperature regulation and stability maintenance provided by the feedback mechanism in the closed-loop system.

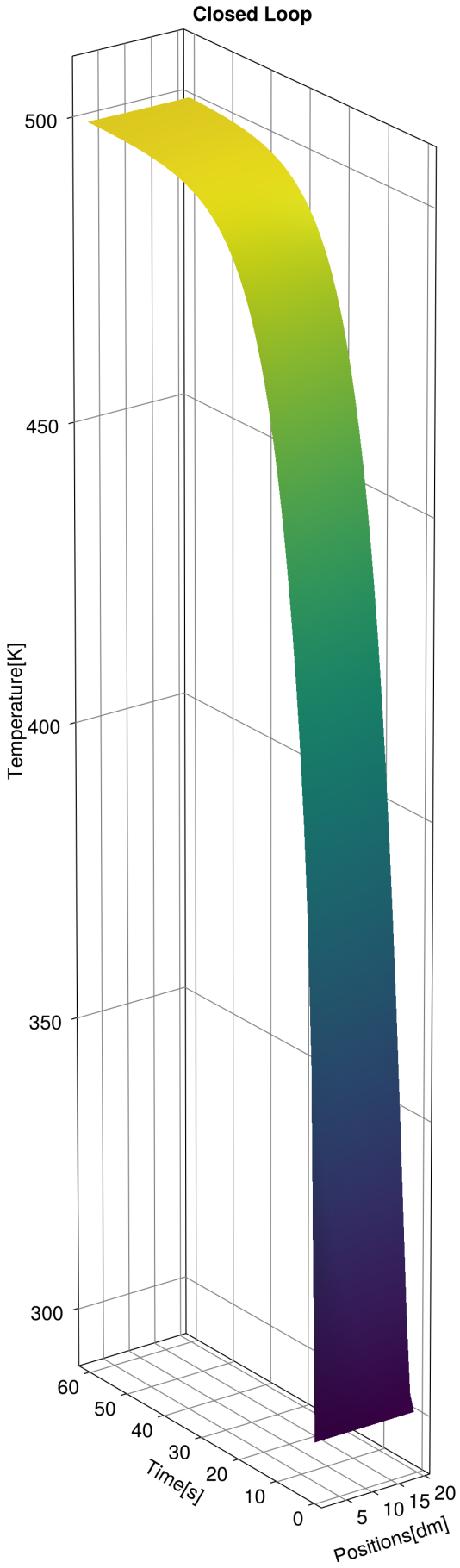

Figure 5.5: Temperature Distribution Over Time in the Closed Loop Heat Conduction System

Nelder-Mead Method with Closed-Loop Control

In the closed-loop system, we utilize a P-Controller to dynamically manage the heat input along the rod. The P-controller modifies the heat input depending on the variation between the measured temperature and the target reference temperature of 500 K. However, for the controller to operate efficiently, it is essential to adjust two key parameters: the thermal conductivity λ and the controller gain K_p.

To accomplish this, the Nelder-Mead optimization method is to determine the true values for these parameters, which are referenced at $\lambda = 50$ and $K_p = 200$. The objective is to decrease the error between the system-measured temperatures and the desired reference temperature by optimizing λ and K_p.

This method is utilized in the code to optimize both λ and K_p for the closed-loop system. The initial values for these parameters are assumed to be $\lambda = 20$ and $K_p = 100$. The optimization process iteratively modifies these values, reducing the system error with each iteration.

```
# 20.0 is the starting guess for the first parameter (λ) and 100.0 for the
    second parameter (Kp).
  initial_λ =  [20.0, 100.0]
# Perform the optimization using the Nelder-Mead method
    result = optimize(loss, initial_λ, NelderMead())
# Extract the optimal values of λ that minimize the loss function
    optimal_λ = Optim.minimizer(result)

```

Code 5: Julia's implementation of the Nelder-Mead Method with closed loop

Figure 5.6 illustrates that the blue curve indicates the optimization of the thermal conductivity λ, and the red curve shows the adjustment of the controller gain K_p. As the optimization advances, both parameters move closer to their true values of $\lambda = 50$ and $K_p = 200$ across sequential iterations. The stability of the system and temperature control depend on these Final values. The successful convergence of λ and K_p to their correct values shows that the Nelder-Mead method effectively minimizes the loss function and optimizes the performance of the controller.

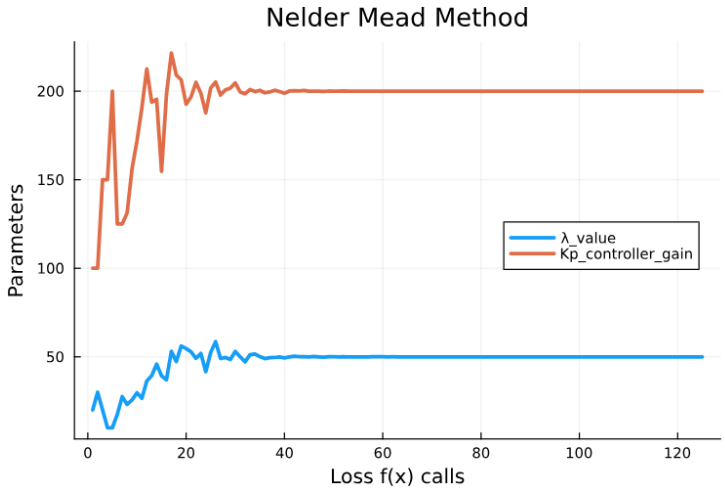

Figure 5.6: Nelder Mead Method with closed loop

Conjugate Gradient Method with Closed-Loop Control

The Conjugate Gradient Method to optimize the thermal conductivity λ and the controller gain K_p in the closed-loop system. Our objective, similar to the previous optimization method, is to minimize the error between the system's actual temperature and the desired target temperature of 500 K. The algorithm aims to converge towards the true values of $\lambda = 50$ and $K_p = 200$, which serve as reference points.

The Conjugate Gradient Method is well-suited for optimization problems involving smooth loss functions and large parameter spaces. It efficiently explores the optimal parameters using gradient information, and the line search approach ensures effective optimization progress.

Code 6 below utilizes the Conjugate Gradient Method in Julia. The **Optimization.AutoForwardDiff()** function is used to calculate gradients for the loss function automatically, ensuring that the optimization method can access the required derivative information to direct the optimization procedure.

```
 # Set the initial guess for the parameter vector λ (two parameters: [10.0, 100.0])
    initial_λ =  [10.0, 100.0]
# AutoForwardDiff automatically computes the gradients during optimization.
    adtype = Optimization.AutoForwardDiff()
# The function' loss(x)' computes the loss, and 'adtype' specifies the type of automatic differentiation.
    optf = Optimization.OptimizationFunction((x, initial_λ) -> loss(x), adtype)
# Create an optimization problem with the objective function and initial parameter values
    optprob = Optimization.OptimizationProblem(optf, initial_λ)
# Solve the optimization problem using the Conjugate Gradient method
# ConjugateGradient is used for the optimization algorithm, and BackTracking is the line search method.
     result_ode = Optimization.solve(optprob, Optim.ConjugateGradient(linesearch=LineSearches.BackTracking()), maxiters = 20)

```

Code 6: Julia's implementation of the Conjugate Gradient Method to estimate λ and K_p.

In Figure 5.7, the optimization process of λ (blue curve) and K_p (red curve) over 20 iterations is presented. Initially, the Conjugate Gradient Method explores various values for both parameters, leading to fluctuations in the first few iterations. However, after around 5 iterations, both λ and K_p start to converge towards their true values of $\lambda = 50$ and $K_p = 200$.

The Conjugate Gradient Method rapidly stabilizes around these values, effectively minimizing the system error and ensuring efficient temperature regulation by the feedback controller. By the 10^{th} iteration, the algorithm essentially identifies the optimal values for both parameters, as evidenced by the leveling off of both curves.

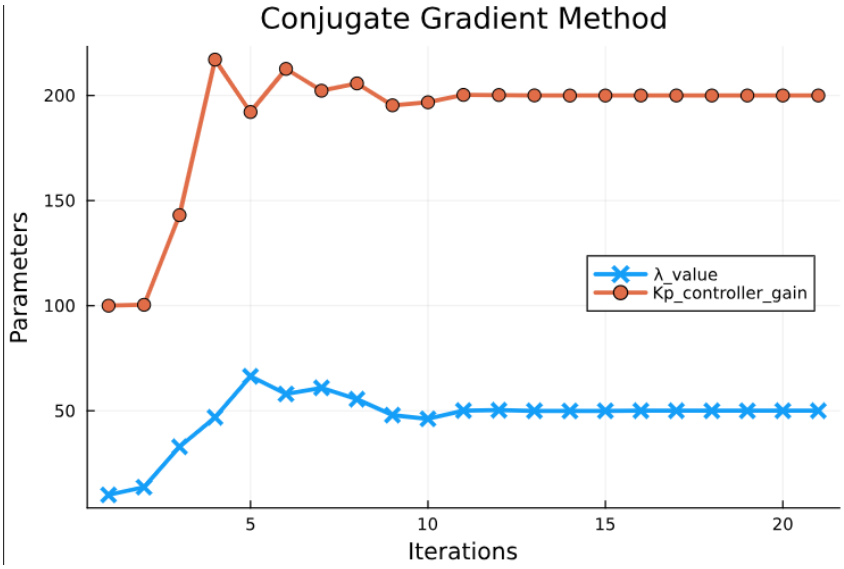

Figure 5.7: Conjugate Gradient Method with closed loop

While the Nelder-Mead and Conjugate Gradient methods were effective, Newton's Method offers a faster convergence by integrating both gradient and Hessian information which is implemented in the upcoming section.

Newton's Method with Closed-Loop Control

Newton's Method is utilized for optimizing the parameters of the closed-loop system, specifically the thermal conductivity λ and the controller gain K_p. Similar to the other optimization methods (Nelder-Mead and Conjugate Gradient), the objective is to minimize the error between the system-measured temperatures and the desired target temperature of 500 K. The true values of $\lambda = 50$ and $K_p = 200$ are used as the reference for the optimization process.

Newton's Method integrates both the gradient (first-order derivative) and the Hessian (second-order derivative), which captures the curvature of the loss function. This allows for more informed and rapid adjustments to the parameters, improving convergence speed and precision.

Below is Code 7 showing how Newton's Method is used in Julia programming to optimize λ and K_p. The **Optimization.AutoForwardDiff()** function calculates the gradients automatically, and the Newton optimizer adjusts the parameters using both the gradient and Hessian.

```
 # Set the initial guess for the parameter vector λ (two parameters: [10.0, 150.0])
   initial_λ =  [10.0, 150.0]
# Define the type of automatic differentiation to be used for gradient and Hessian calculation.

    adtype = Optimization.AutoForwardDiff()
    # OptimizationFunction takes the parameter values (x) and the initial guess (initial_λ) to calculate the loss.
# The loss function (loss(x)) computes the error based on the given values of λ.
    optf = Optimization.OptimizationFunction((x, initial_λ) -> loss(x), adtype)
# Create the optimization problem using the defined objective function and initial values.
    optprob = Optimization.OptimizationProblem(optf, initial_λ)
# Optimization.solve runs the optimization using Newton's method.
# Newton's method uses both the gradient and Hessian of the loss function for faster convergence.
    result_ode = Optimization.solve(optprob, Optim.Newton(), maxiters = 25)


```

Code 7: Julia implementation of the Newton's Method to estimate λ and K_p.

In Figure 5.8, the optimization process of thermal conductivity λ and controller gain K_p is illustrated over 25 iterations using Newton's Method. Initially, the algorithm explores various values for both parameters, resulting in fluctuations during the early iterations. However, after around 15 iterations, both parameters start to stabilize, with λ approaching 50 and K_p approaching 200.

Upon reaching the 25th iteration, both parameters reach their optimal values, effectively minimizing the error between the measured and target temperatures. This efficient convergence underscores the effectiveness of Newton's Method in optimizing closed-loop systems.

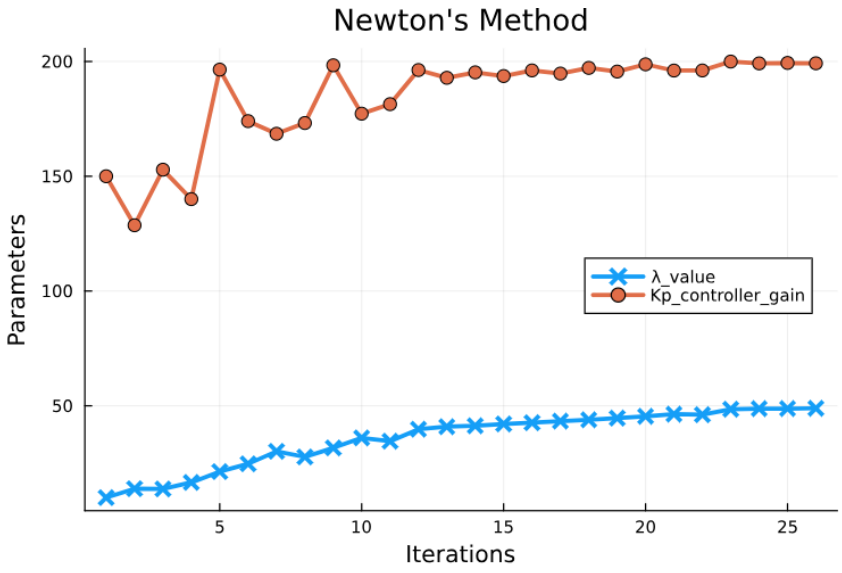

Figure 5.8: Newton's Method with closed loop

5.3 Evaluating Sensor Requirements for Robust Parameter Estimating in Noisy Environments

After achieving parameter identification through the closed-loop system using the Conjugate Gradient method, the results are implemented in a real-world scenario accounting for external factor Noise. An analysis of how the number and placement of sensors on the rod affect the accuracy of estimates in both noiseless (simulation) and noisy (real-world) conditions.

Table 5.1 presents the results of applying the Conjugate Gradient Method to estimate two critical parameters in a controlled heat conduction system: thermal conductivity λ and controller gain K_p. These parameters are critical for achieving the desired temperature using a P-controller.

In the case of the X1 setup, where only one sensor is positioned at the beginning of the rod, the results could have been more accurate. In the absence of noise, the predicted value of λ deviated significantly from the true value of 50, indicating that a single sensor alone cannot capture the complete heat distribution along the rod. Moreover, the controller gain K_p also demonstrated substantial deviations from the true value of 200. The errors escalated with the introduction of noise, as evidenced by the increase in MAE. A single sensor is inadequate for parameter estimation in noisy conditions.

The results were unsatisfactory in the (X1:X2) or (X2:X3) setup, where two sensors are positioned closely. In the absence of noise, the estimated value of λ was approximately 14.25, significantly deviating from the true value. The closeness of the sensors restricts the ability to accurately capture the system dynamics, which results in an inaccurate estimation. Noise further worsened the performance, leading to increased deviations in both λ and K_p and a corresponding rise in MAE. This indicates that implementing two closely placed sensors offers inadequate information for precise estimation. Conversely, the X1, X11, and X21 configurations, utilizing three sensors distributed more widely along the rod's length, yielded significantly more precise estimates. In the noiseless scenario, the estimated λ was almost equal to the true value 50. They suggest that this arrangement effectively captures the temperature distribution across the system. Even in the presence of noise, the estimated λ remained close to 49.8, and K_p stayed near 200, showcasing the resilience of this setup. The broader sensor distribution enables a more comprehensive understanding of the system's thermal behavior, making the estimation process less vulnerable to noise.

The instances above demonstrate that setups with fewer sensors, such as X1 (one sensor) or X1:X2 (two closely positioned sensors), produce imprecise parameter estimates and are highly vulnerable to interference. On the other hand, the X1, X11, and X21 arrangement, comprising three well-distributed sensors, produces stable and accurate estimates for both λ and K_p, even in noisy surroundings. Hence, to attain accurate parameter estimation, a minimum of three strategically positioned sensors is essential to counteract the impact of interference and capture the complete system dynamics.

Table 5.1: Closed-Loop System: Conjugate Gradient Method for estimation of parameters

Measure points	Time (s)	MAE $[K]$	λ in $\left[\frac{W}{mK}\right]$	K_p	MAE (with noise)$[K]$	λ in $\left[\frac{W}{mK}\right]$	K_p
X 1 : X 21	20.11543	1.04×10^{-5}	49.99985998	199.9998618	2.349974256	49.83525471	200.7222372
X 2 : X 21	20.551556	2.19×10^{-6}	50.00004306	200.0000217	2.403022717	50.02144977	200.323863
X 3 : X 21	20.221792	3.48×10^{-6}	50.00007272	200.0000399	2.411352075	51.1359187	201.0341544
X 4 : X 21	19.580604	4.96×10^{-7}	50.0000019	199.9999968	2.417496022	49.31760609	199.6002643
X 5 : X 21	20.999313	2.68×10^{-6}	50.00006229	200.0000246	2.426476791	49.95834335	200.0742388
X 14 : X 21	19.215468	1.25×10^{-5}	50.00028055	200.0000734	2.411644716	48.02611939	199.1618335
X 15 : X 21	19.477079	1.59×10^{-4}	50.00347498	200.0008544	2.400440299	49.75906891	200.1208606
X 16 : X 21	20.62549	1.38×10^{-5}	50.00027515	200.000027	2.330221995	48.89389136	200.6593061
X 17 : X 21	19.243875	7.75×10^{-6}	50.00014232	200.0000023	2.335251992	52.64762744	200.4643719
X 18 : X 21	19.249862	1.55×10^{-5}	50.00031236	200.00005	2.245583212	51.94664404	198.1607739
X 19 : X 21	19.502629	3.17×10^{-6}	50.00006257	200.000008	2.47915045	50.63808739	199.9033825
X 20 : X 21	20.170927	1.62×10^{-5}	50.00032332	200.0000529	2.468650563	51.31351883	200.9390228
X 21	19.223894	1.58×10^{-5}	50.00031957	200.0000795	2.260516051	48.67563634	205.846483
X 11	19.375773	7.46×10^{-6}	50.00033134	200.000132	2.461752259	50.37490626	201.6564933
X 1	18.823422	1.6783	14.68337539	101.742393	2.719456286	11.25301583	83.12279713
X 1 : X 2	20.799352	1.7613	14.25413809	102.0198844	3.419322219	14.44970031	101.9555105
X 2 : X 3	19.31116	1.5851	13.40272712	103.0963693	2.704614252	13.63460986	102.9722658
X 3 : X 4	21.435818	1.2807	17.92854517	128.6846719	2.811237623	20.21791658	136.310837
X 4 : X 5	20.682573	1.5634×10^{-4}	49.99357138	199.992926	2.451405907	25.24694773	157.5962782
X 1 , X 11, X 21	19.888524	1.76×10^{-6}	49.99998058	199.9999723	2.532684386	50.62468892	201.0914334

5.4 Conclusion

This thesis successfully demonstrated the effectiveness of parameter identification in a one-dimensional heat conduction system using both open-loop and closed-loop control approaches. The open-loop system, which is useful for collecting baseline temperature data, was limited in maintaining the desired temperature due to the lack of feedback. In contrast, the closed-loop system, integrated with a P-Controller, consistently achieved the target temperature with superior adaptability and precision, even under changing conditions.

This study addressed the challenge of estimating key parameters, such as thermal conductivity (λ) and controller gain (Kp), by using three optimization methods: Nelder-Mead, Conjugate Gradient, and Newton's Method. Among these methods, the Conjugate Gradient Method was particularly effective in balancing speed and accuracy, making it adaptable for real-world applications.

The Conjugate Gradient Method utilized gradient information to achieve faster convergence without the complexity of second-order derivative calculations required by Newton's Method. Its efficiency and straightforward implementation make it highly suitable for applications with moderate computational resources and accessible gradient information. Therefore, it proved to be the most practical optimization method for parameter estimation, particularly in cases where rapid and accurate identification is essential.

Although simpler and not dependent on gradients, Nelder-Mead required more iterations to reach a solution. It is more suitable for non-smooth cost functions or situations where gradient information is difficult to obtain. However, its slower convergence limited its application in more demanding scenarios. On the other hand, Newton's Method used gradient and second-order derivative (Hessian) information, offering the highest precision and the fewest iterations. While it provided the best performance in terms of speed and accuracy, its reliance on complex derivative calculations makes it more computationally intensive.

The choice of optimization method largely depends on the specific application requirements—whether the priority is simplicity, computational speed, or precision.

Furthermore, this study highlighted the significance of sensor configuration in parameter estimation. It was observed that increasing the number of sensors above three, especially in noisy environments, significantly improved the accuracy of parameter estimates. The recommended configuration of more than three sensors offers a practical, cost-effective solution for ensuring robust parameter identification in industrial applications.

In conclusion, this thesis demonstrated that closed-loop systems, combined with gradient-based optimization methods like the Conjugate Gradient, provide a powerful approach to parameter identification in heat conduction systems. The specific problem requirements should guide the selection of an appropriate optimization method, whether they demand computational simplicity, rapid convergence, or high precision. Future research could explore hybrid optimization techniques or apply these methods to more complex systems to enhance performance in challenging environments.

Bibliography

[1] Chandrasyah. *nderstanding heat transfer: a fundamental phenomenon in engineering and nature.* 28 July. 2023. URL: https://medium.com/@chandrasyah999/understanding-heat-transfer-a-fundamental-phenomenon-in-engineering-and-nature-fe267a25f8ec (cited on page 1).

[2] Piranha. *What is Industrial Heat Transfer*. 22 Feb. 2022. URL: https://newsome.ltd.uk/what-is-industrial-heat-transfer/ (cited on page 1).

[3] FRANK P. INCROPERA. , *DeWitt, D. P., Bergman, T. L., Lavine, A. S. Fundamentals of Heat and Mass Transfer, 96.* URL: https://hyominsite.wordpress.com/wp-content/uploads/2015/03/fundamentals-of-heat-and-mass-transfer-6th-edition.pdf (cited on page 1).

[4] Admin BYJUS. *What Is Heat Transfer? Conduction, Convection, Radiation and FAQs.* Accessed 19 Sept. 2024. 28 Aug. 2018. URL: https://byjus.com/physics/heat-transfer-conduction-convection-and-radiation/ (cited on page 1).

[5] A. A. Minea. *Advances in industrial heat transfer,1-3.* 2013. URL: https://api.pageplace.de/preview/DT0400.9781439899083_A23974416/preview-9781439899083_A23974416.pdf (cited on page 1).

[6] D. Sterian. , *Alina-Ioana, C. Mathematical and numerical modeling of inverse heat conduction problem.* 2014. URL: https://doi.org/10.13111/2066-8201.2014.6.4.3 (cited on page 1).

[7] A. V. Wouwer. , *Point, N., Porteman, S., Remy, M. An approach to the selection of optimal sensor locations in distributed parameter systems, 298.* 2000. URL: https://www.wellesu.com/10.1016/s0959-1524(99)00048-7 (cited on page 1).

[8] Cornelis Vuik et al. *Numerical Methods for Ordinary Differential Equations, 119-120.* TU Delft OPEN Textbook. 2023. DOI: 10.5074/t.2023.001. URL: https://doi.org/10.5074/t.2023.001 (cited on page 3).

[9] Admin. *Solving the heat equation with Neumann boundary conditions.* URL: https://math.stackexchange.com/questions/3928125/solving-the-heat-equation-with-neumann-boundary-conditions (cited on page 4).

[10] F. (n.d.). Van Der Plas; Mikołaj Bochenski. *Heat equation with Neumann boundary conditions*. URL: https://elearning.rwu.de/pluginfile.php/221542/mod_resource/content/2/neumann_numerical.jl.html (cited on pages 5, 8).

[11] Admin. *ODE Solvers · DifferentialEquations.JL. (n.d.).* URL: https://docs.sciml.ai/DiffEqDocs/stable/solvers/ode_solve/ (cited on page 9).

[12] Wikipedia contributors. *Runge–Kutta methods*. Accessed 19 Sept. 2024. 2024, August 27. URL: https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods (cited on page 9).

[13] Admin. *Thermocouple accuracies*. URL: https://www.thermocoupleinfo.com/thermocouple-accuracies.htm (cited on page 10).

[14] Admin. *Gaussian$_N$oise*.. URL: https://www.sfu.ca/sonic-studio-webdav/handbook/Gaussian_Noise.html (cited on page 10).

[15] Wikipedia contributors. *Open-loop controller*. 13 Mar. 2024. URL: https://en.wikipedia.org/wiki/Open-loop_controller (cited on page 11).

[16] Thdata. *Closed-Loop vs. Open-Loop Production Control: Examples and Differences*. URL: https://planeus-solutions.com/blog/en/closed-loop-vs-open-loop-production-control-system/ (cited on page 12).

[17] Wikipedia contributors. *Closed-loop controller*. 15 Jan. 2024. URL: https://en.wikipedia.org/wiki/Closed-loop_controller (cited on page 13).

[18] Admin. *Proportional (P) controller*. URL: https://x-engineer.org/proportional-controller/ (cited on page 13).

[19] GeeksforGeeks. *Proportional controller in control system. GeeksforGeeks*. 20 Oct. 2023. URL: https://www.geeksforgeeks.org/proportional-controller-in-control-system/ (cited on page 14).

[20] Xu K. *Darve, E. Machine learning for inverse problems in computational engineering*. URL: https://kailaix.github.io/ADCMESlides/2020_10_01.pdf (cited on page 16).

[21] DataRobot. DataRobot. *Introduction to loss functions*. 15 FEb. 2024. URL: https://www.datacamp.com/tutorial/loss-function-in-machine-learning (cited on page 18).

[22] Wikipedia contributors. *Mean squared error*. 11 June. 2024. URL: https://en.wikipedia.org/wiki/Mean_squared_error (cited on page 18).

[23] R. Alake. *Loss functions in machine learning explained*. 24 Nov. 2023. URL: https://www.datacamp.com/tutorial/loss-function-in-machine-learning (cited on page 19).

[24] Wikipedia contributors. *Mean absolute error*. 2 April. 2024. URL: https://en.wikipedia.org/wiki/Mean_absolute_error (cited on page 19).

[25] J. Brownlee. *How to choose an optimization Algorithm*. 11 Oct. 2021. URL: https://machinelearningmastery.com/tour-of-optimization-algorithms/ (cited on page 20).

[26] Wikipedia contributors. *Nelder–Mead method*. 18 Sep. 2024. URL: https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method (cited on page 20).

[27] S. Singer. , *Nelder, J. Nelder-Mead algorithm*. 2009. URL: https://doi.org/10.4249/scholarpedia.2928 (cited on page 21).

[28] Magnus R. Hestenes and Eduard Stiefel. *Methods of Conjugate Gradients for Solving Linear Systems*. 6, December 1952. URL: https://nvlpubs.nist.gov/nistpubs/jres/049/jresv49n6p409_A1b.pdf (cited on page 21).

[29] Wikipedia. Hestenes Stiefel. *Conjugate gradient method.* 2021. URL: https://pages.stat.wisc.edu/~wahba/stat860public/pdf1/cj.pdf (cited on page 22).

[30] Wikipedia contributors. *Newton's method in optimization.* 12 Sep. 2024. URL: https://en.wikipedia.org/wiki/Newton%27s_method_in_optimization (cited on page 22).

[31] Admin. *Newton's Method.* URL: https://acme.byu.edu/00000180-6940-dc15-abb6-f9df79240001/newtons-method (cited on page 22).

[32] Wikipedia contributors. *Automatic differentiation.* 24 Sep. 2024. URL: https://en.wikipedia.org/wiki/Automatic_differentiation (cited on page 23).