Hochschule Ravensburg-Weingarten
Faculty for Electrical Engineering and Computer Science

HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Optimal Control of the heat equation in Julia

Bachelor Thesis

submitted for the degree of

Bachelor of Electrical Engineering and Information Technology

by
Abdelhady Abdelmagid
Matriculation Nr.: 27893

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

ABSTRACT

Optimal control of systems described using partial differential equations is a field of interest in research and industry. The heat equation, a partial differential equation, describes various heating and cooling processes. It is usually accompanied by constraints on the input signals, temperature through the body or both, which motivates applying optimal control in such processes.

In this thesis, a solution to the optimal boundary control of the heat equation is presented. The heat equation is numerically approximated using Finite difference method. The optimal control problem is formulated using a direct method into a non linear program. JuMP, a mathematical modelling language built on Julia is used to solve the problem. Simulation results are presented for a one dimensional rod and a two dimensional plate.

Contents

Chapter 1

Problem statement

In this chapter, we start by introducing the motivation to our work in Section 1.1. In Section 1.2, a brief introduction of the heat equation is given. A problem statement and the objectives are given in Section 1.3.

1.1 Motivation

Optimal control theory has been a major topic in research and industry, as it controls a process in the most efficient way according to a criterion specified. Different numerical approaches are available for solving such problems using mathematical modelling frameworks, such as JuMP [7], NLOptControl [8], CasADi [1] and various other frameworks. With the computational power available nowadays, numerical solutions became feasible to use with systems described using partial differential equations (PDE).

The boundary control of the heat equation arises in heating and cooling applications as in [2, 4]. The heat equation is also known to be one of the basic partial differential equation, which makes it suitable for researching optimal control theory.

The boundary input heat equation is usually accompanied by constraints on the actuator's input, this increases the problem's size to an already large scaled problem. Different approaches were introduced to solve a constrained optimal control problem (OCP) of heat equation. In [23], a method is introduced to reformulate the problem into an unconstrained optimization problem. However, the presence of mathematical modelling packages such as JuMP [7] and NLOptControl [8] motivates this work of solving the OCP of heat equation with boundary input constraints using a direct method [19].

1.2 Heat equation

The heat equation is a partial differential equation,that describes temperature distribution $\theta(x,t)$ in space and time. The linear heat equation in three dimensions is noted as

$$\frac{\partial\theta}{\partial t} = \alpha\left(\frac{\partial^2\theta}{\partial x^2} + \frac{\partial^2\theta}{\partial y^2} + \frac{\partial^2\theta}{\partial z^2}\right). \tag{1.1}$$

The optimal boundary control of heat equation is considered for one dimensional and two dimensional cases.

One dimensional heat equation

As presented in Figure 1.1, a one dimensional body $\Omega = (0, L)$, with a heating source placed on the left side and a temperature sensor placed on the right side of the body in the time $t \in (0, t_f)$ is considered.

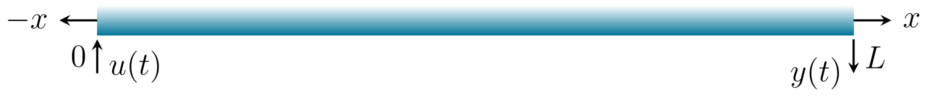


Figure 1.1: A one dimensional rod Ω of length L with the input source on left side and output source on the right side.

From Equation (1.1), the one dimensional case of the heat equation is noted as

$$\frac{\partial\theta}{\partial t} = \alpha\frac{\partial^2\theta}{\partial x^2}, \tag{1.2}$$

with $x \in \Omega$. The initial and boundary conditions are also needed to define the function. Assuming a constant initial temperature $\theta(x,0) = \theta_0$.

The Neumann boundary condition describes the normal flux along the boundary [17]. Neumann boundary condition is noted as

$$\lambda\nabla\theta(x,t)\cdot\vec{n} = 0, \tag{1.3}$$

where n is a vector pointing to x direction. The boundary conditions for input placed on the left side and measurements taken on the right

$$\begin{aligned} -\lambda\nabla\theta(0,t) &= u(t), \\ \lambda\nabla\theta(L,t) &= 0. \end{aligned} \tag{1.4}$$

Two dimensional heat equation

As shown in Figure 1.2, a two dimensional body $\Omega = (0, L) \times (0, W)$ in time interval $t \in (0, t_f)$ is studied.

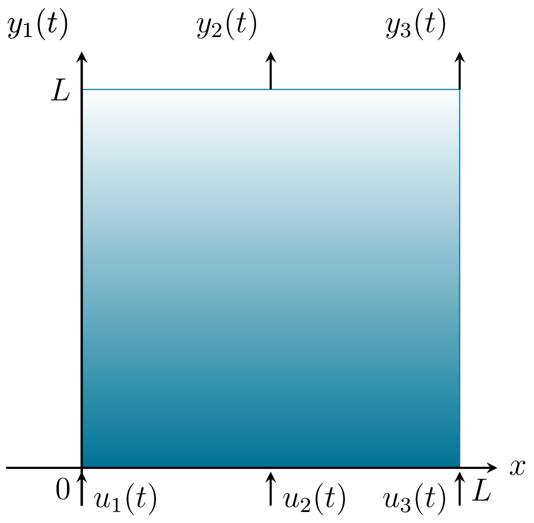


Figure 1.2: A two dimensional plate Ω of length L and width L with several input sources on lower side and several measurements on the upper side.

From the Equation (1.1), the two dimensional heat equation is noted as

$$\frac{\partial \theta}{\partial t} = \alpha \left(\frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2} \right), \tag{1.5}$$

with $x, y \in \Omega$. The initial temperature is $\theta\left(x, y, 0\right) = \theta_0$.

The Neumann boundary conditions in Equation (1.3) is also applied for the two dimensional case, where $\nabla \theta, n \in R^2$. The boundary condition for the two dimensional space is noted as

$$\lambda \begin{bmatrix} \dfrac{\partial \theta}{\partial x} \\ \dfrac{\partial \theta}{\partial y} \end{bmatrix} \cdot \vec{n} = 0, \tag{1.6}$$

where $\vec{n}$ represents the sides of the space. The possible values for $\vec{n}$ are

$$\begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{1.7}$$

The outcome of Equation (1.6) is four boundary conditions. The arrangement of the inputs applied on the lower side of the body, the boundary conditions are

$$\begin{aligned} -\lambda \frac{\partial \theta(0,y,t)}{\partial x} &= 0, \\ \lambda \frac{\partial \theta(L,y,t)}{\partial x} &= 0, \\ -\lambda \frac{\partial \theta(x,0,t)}{\partial y} &= u(x,t), \\ \lambda \frac{\partial \theta(x,W,t)}{\partial y} &= 0. \end{aligned} \tag{1.8}$$

1.3 Problem statement

The interest of this work is to calculate the optimal input $u^*(t)$ of a boundary input two dimensional heat equation, where the body is heated to a given temperature with a bounded input signal. A one dimensional rod is first to be studied with a heating source on the left side and output measured on the right side. The controller is then adjusted to implement a trajectory for several input sources on one side and several measurements on the other side of the body. The final output is an optimal boundary control of the one dimensional and two dimensional heat equation in JuMP.

Chapter 2

Numerical Optimal Control

In this chapter, we introduce the topics optimal control and numerical optimization. We start by introducing the state space representation of system dynamics in Section 2.1. This is followed by a brief introduction of Finite difference method. In Section 2.2, an approximation of the heat equation using finite difference method is presented for the one dimensional and two dimensional bodies with boundary input, to represent the heat equation in state space. The stability, controllability and observability are tested for the propsed approximation in Section 2.2. In Section 2.3, the general form of a cost functional and the linear quadratic regulators are introduced. In Section 2.4, Numerical optimization is introduced briefly. In Section 2.5, direct collocation method is introduced. Finally, the formulated OCP of the heat equation is presented in Section 2.6.

2.1 State space representation

The formulation of a control problem requires the presence of a mathematical model describing the system dynamics. The system dynamics are represented in state space representation, when used in OCPs. This form of representation reduces the order of the differential equations of the system dynamics to first order differential equations. The states of a system are defined by a vector $x(t) \in \mathbb{R}^n$, with n as the dimension of the system. An initial value of the vector $x(t_0)$, in addition to the control input $u(t) \in \mathbb{R}^m$,where m is the number of input signals, are used to determine the states vectors values over time for $t > 0$. The system dynamics is described by

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{2.1}$$

with matrix $A \in \mathbb{R}^{n\times n}$ and matrix $B \in \mathbb{R}^{n\times m}$ for a linear, time invariant system.

The output equations represent the states that can actually be measured by the system. For a linear, time invariant system, the output equation has the following form

$$y(t) = Cx(t) + Du(t), \tag{2.2}$$

with matrix $C \in \mathbb{R}^{q \times n}$ and matrix $D \in \mathbb{R}^{q \times m}$, where $y(t) \in \mathbb{R}^q$. The matrix D is zero in the cases studied. Both, the system equation together with the output equation, fully represent a model.

If a system is described with the second order differential equation

$$\ddot{y}(t) + a\dot{y}(t) + by(t) = u(t). \tag{2.3}$$

We rename the variables in Equation (2.3) to get the state vector $x(t)$,where $x_1(t) = y(t)$ and $x_2(t) = \dot{y}(t) = \dot{x}_1(t)$. The second order differential equation is reduced into first order differential equations, which are noted as

$$\begin{aligned} \dot{x}_1(t) &= \dot{y}(t) = x_2(t), \\ \dot{x}_2(t) &= \ddot{y}(t) = -ax_2(t) - bx_1(t) + u(t). \end{aligned} \tag{2.4}$$

From Equation (2.4), The matrices A and B are

$$A = \begin{pmatrix} 0 & 1 \\ -b & -a \end{pmatrix}, B = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.5}$$

The output matrix is based on the sensors used and their functionality. If the sensors are arranged to measure the first state of the system, the output matrix is

$$C = \begin{pmatrix} 1 & 0 \end{pmatrix}. \tag{2.6}$$

The system dynamics are described by

$$\begin{aligned} \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} &= \begin{pmatrix} 0 & 1 \\ -b & -a \end{pmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u(t), \\ y(t) &= \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}. \end{aligned} \tag{2.7}$$

An example is the mechanical oscillator explained in [20]. The system is described by

$$\ddot{y}(t) + 2d\omega_0\dot{y}(t) + \omega_0^2 y(t) = K\omega_0^2 u(t). \tag{2.8}$$

The system in state space representation is noted as

$$\begin{pmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\omega_0^2 & -2d\omega_0 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} + \begin{pmatrix} 0 \\ K\omega_0^2 \end{pmatrix} u(t). \tag{2.9}$$

In Chapter 3, this example is used to demonstrate the software tools JuMP [7] and NLOptControl [8].

Finite difference Method

Finite difference methods (FDM) are used in the approximation of differential equations to difference equations, so that they can be solved numerically. It is also used to discretize PDEs, where changes in each of the discretized spaces can be studied with respect to time. The equations mentioned in this section follows [17].

In OCPs, the system dynamics are usually represented in state space as a number of first order differential equations. FDMs could be used in order to describe a PDE in the form of difference equations that describes a system and can be used to solve the OCP numerically. The error between the numerical and exact value of the derivative is called truncation error and it arises from the fact that the FDM is obtained from the approximation of the taylor series. Considering a one dimensional equation, the approximation is related to the definition of the derivative of a smooth function u at point $x \in \mathbb{R}$, where the derivative of u is noted as

$$u^{'}(x) = \lim_{h \to 0} \frac{u(x+h) - u(x)}{h}. \tag{2.10}$$

The smaller the value of h is, the better the value of this approximation. This means that the error of the numerical result decreases with reducing the value of h towards 0.

By applying Taylor series, for any value of $h > 0$, for the first derivative

$$u(x+h) = u(x) + hu^{'}(x) + \frac{1}{2}u^{''}(\xi)h, \tag{2.11}$$

in which ξ is any value between $[x, x+h]$, the truncation error in Equation (2.11) is proportional to the step size h and is denoted as $O(h)$.

For the second derivative, we consider the taylor series again

$$\begin{aligned} u(x+h) &= u(x) + hu^{'}(x) + \frac{h^2}{2}u^{''}(x) + \frac{h^3}{6}u^{'''}(x) + O(h^4), \\ u(x-h) &= u(x) - hu^{'}(x) + \frac{h^2}{2}u^{''}(x) - \frac{h^3}{6}u^{'''}(x) + O(h^4). \end{aligned} \tag{2.12}$$

From adding both parts of the Equation (2.12), the difference equation of the second derivative is noted as

$$u^{''}(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + O(h^2). \tag{2.13}$$

The center difference approximation of the second order dervative has a truncation error proportional to h^2. As shown in Figure 2.1, a one dimensional rod is discretized into N nodes.

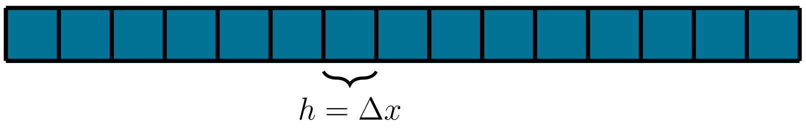


Figure 2.1: The discretization of a one dimensional rod into a number of nodes $N = 16$.

The discretization of a two dimensional space is presented in Figure 2.2.

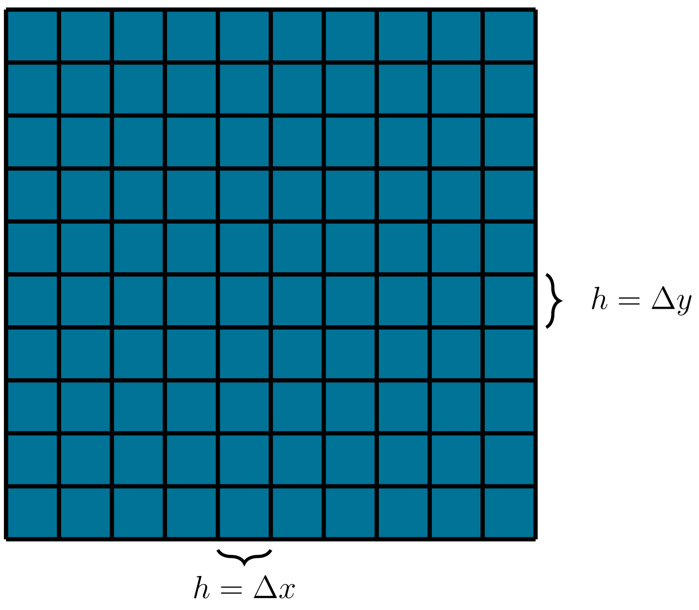


Figure 2.2: The discretization of a two dimensional plate into a number of nodes $N_x = N_y = 11$.

2.2 Heat equation in State space representation

One dimensional heat equation

Equation (1.2) shows the heat equation in a one dimensional space. To obtain a numerical approximation of the heat equation, the equation is discretized in space and time. For the spatial discretization, center difference approximation noted in Equation (2.13) is applied. The heat equation in discretized space is

$$\frac{\partial \theta \left(x_i, t\right)}{\partial t} = \alpha \left(\frac{\theta_{i+1} - 2\theta_i + \theta_{i-1}}{\Delta x^2}\right), \tag{2.14}$$

where the node arrangement is presented in Figure 2.3.

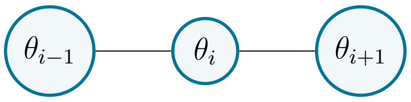


Figure 2.3: The visualisation of the arrangement of the nodes surrounding node i.

In Equation (2.14), $\Delta x = h$ is the step size between two discretization points. Equation (2.14) shows the change in temperature at any discretized space x_i in the one dimensional body considered. The boundary conditions are also discretized. From Equation (1.4), the boundary condition for both ends of the rod are noted as

$$\begin{aligned} \theta_1 - \theta_{-1} &= -\frac{2\Delta x}{\lambda} u(t), \\ \theta_{N+1} - \theta_{N-1} &= 0, \end{aligned} \tag{2.15}$$

where θ_{-1} and θ_{N+1} are imaginary points used to calculate the change at $x_i = 0$ and $x_i = N$, where N is the number of discretization points.

From Equation (2.14), the system matrix is obtained. The system matrix is a tri-diagonal sparse matrix, with non-zero entries in the first three diagonals in the matrix. The system matrix has the form

$$A = \frac{\alpha}{\Delta x^2} \begin{pmatrix} -2 & 2 & 0 & \cdots & \cdots & 0 \\ 1 & -2 & 1 & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \cdots & \vdots \\ \vdots & 0 & \cdots & 1 & -2 & 1 \\ 0 & \cdots & \cdots & \cdots & 2 & -2 \end{pmatrix}. \tag{2.16}$$

The first entry of the upper and the last entry of the lower secondary diagonal arises from the discretization of boundary conditions in Equation (2.15).

From Equation (2.15), the input matrix is formed. There is one input source applied on the left side of the rod, which means that there is only one entry at the beginning of the input matrix. The input matrix is written as

$$B = \frac{2}{c\rho\Delta x}\begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix}. \tag{2.17}$$

Equation (2.11) shows the approximation of first order derivatives. At a position x_i at a time instance k with a time step Δt, the heat equation is approximated to

$$\frac{\theta^{k+1}-\theta^{k}}{\Delta t} = A\theta^{k} + Bu^{k}. \tag{2.18}$$

The approximation of the heat equation is forward time center space, which is an explicit method that have conditions for numerical stability. According to [17], the inequality constraint must satisfy

$$\frac{\alpha\Delta t}{\Delta x^2} \leq \frac{1}{2}. \tag{2.19}$$

Two dimensional heat equation

Equation (1.5) represents the two dimensional heat equation. The index i is used for the iteration in the x-direction and j index for iteration in the y-direction. The center difference method in Equation (2.13) is used for the discretization of the temperature change in space. The difference equation for discretization in space is noted as

$$\frac{\partial\theta(i,j,t)}{\partial t} = \alpha\left(\frac{\theta_{i+1,j}-2\theta_{i,j}+\theta_{i-1,j}}{\Delta x^2} + \frac{\theta_{i,j+1}-2\theta_{i,j}+\theta_{i,j-1}}{\Delta y^2}\right). \tag{2.20}$$

The node arrangement used in the approximation is presented in Figure 2.4.

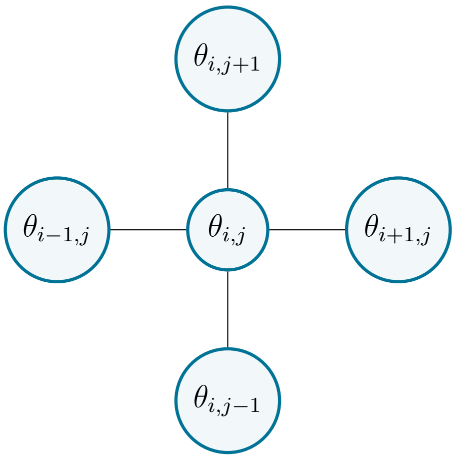


Figure 2.4: The visualisation of the arrangement of the nodes surrounding node (i, j).

If $\Delta x = \Delta y$ in Equation (2.20), they are replaced by a step h. The center difference equation is noted as

$$\frac{\partial \theta(i,j,t)}{\partial t} = \frac{\alpha}{h^2}\left(\theta_{i+1,j} - 4\theta_{i,j} + \theta_{i-1,j} + \theta_{i,j+1} + \theta_{i,j-1}\right). \tag{2.21}$$

The boundary conditions are discretized as done with the one dimensional case. The discretized boundary conditions are written as

$$\begin{aligned} -\lambda \frac{\theta_{1,j}(t) - \theta_{-1,j}(t)}{2h} &= 0, \\ \lambda \frac{\theta_{N+1,j}(t) - \theta_{N-1,j}(t)}{2h} &= 0, \\ -\lambda \frac{\theta_{i,1}(t) - \theta_{i,-1}(t)}{2h} &= u(t), \\ \lambda \frac{\theta_{i,N+1}(t) - \theta_{i,N-1}(t)}{2h} &= 0. \end{aligned} \tag{2.22}$$

The forward difference of the time derivative is noted as

$$\frac{\theta^{k+1} - \theta^k}{\Delta t} = A\theta^k + Bu^k. \tag{2.23}$$

For the numerical stability of the two dimensional equation, the inequality constraint must satisfy

$$\frac{\alpha \Delta t}{h^2} \leq \frac{1}{4}. \tag{2.24}$$

Stability

The system's stability describes the evolution of the system in time. A system is called stable, if the solution is expected to converge to an equilibrium point. If the system is unstable, the solution might diverge to arbitrary values of initial data or input systems.

According to [16], a system is stable, if the eigenvalues of the system ($\lambda_i \leq 0$) for $i = 0, 1, 2 \cdots k, k \leq n$, where n is number of states.

The eigenvalues of the discretized one dimensional heat equations are shown in Figure 2.5.

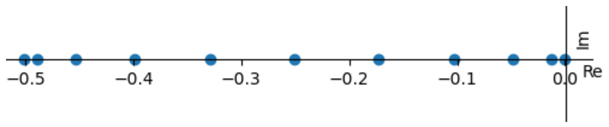


Figure 2.5: Eigenvalues of the one dimensional heat equation with 11 discretization points.

The largest eigenvalue is equal to zero, which means that the one dimensional heat equation is stable. The eigenvalues of the two dimensional heat equation are shown in Figure 2.6.

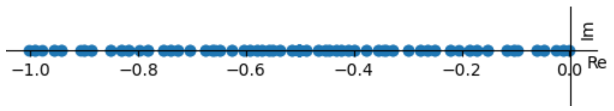


Figure 2.6: Eigenvalues of the two dimensional heat equation with 11 discretization points in each dimension.

Similar to the one dimensional case, the greatest eigenvalue is equal to zero, which shows the stability of the system.

Controllability and observability

Controllability shows the degree of controllability of a system. A fully controllable system, is a system, whose states can be driven from one state to another during a finite time horizon, depending on the control inputs of the system. The concept of Controllability was first introduced in [12].

According to [16], the controllability matrix is noted as

$$E = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix}. \tag{2.25}$$

A system is fully controllable if the rank of matrix E is equivalent to the number of states in the system.

The controllability of the approximation of the heat equation is done by implementing the controllability matrix for various N values. For all tested values from $N = 3$ to $N = 15$, the controllability matrix has full rank. The form of the controllability matrix is similar for all values of N, where the matrix is an upper triangular matrix with values added to the upper side of the matrix only. The test proves that the approximation of the heat equation is fully controllable for the tested values.

Another principle introduced in [12] was the Observability of a system . An observable system is a system where the initial states $x(t_0)$ can be determined by knowing the input of the system $u(t)$ and the output of the system $y(t)$ across a finite time interval $[t_0, t_f]$.

Observability matrix [16] is used to test observability of the system, where the rank of observability matrix G $= n$, where n is the number of states

$$G = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}. \tag{2.26}$$

The observability of the approximation of the heat equation is tested in the same way as the controllability of the matrix. The observability matrix was created for $N = 3$ up to $N = 15$. The matrix had entries in the lower part of the matrix with full rank of the matrix, which proves the observability of the system to the tested values.

2.3 Cost Function

The task of the optimal control problem is to find a signal u^*, that results in an optimal state trajectory x^* with respect to a performance criterion, known as a cost function. The general form of a cost function is noted as

$$J(x(t), u(t), t) = J(t_0, t_f, x(t_0), x(t_f)) + \int_{t_0}^{t_f} \omega(\tau, x(\tau), u(\tau))\, \mathrm{d}\tau. \tag{2.27}$$

There are two parts forming the cost function. The part with the boundary states and inputs, is known in literature as Mayer's form and the part representing an integral along path of trajectory is known as the Lagrange term. The previous cost function is in Bolza form, which is a combination of both terms [14]. The choice of which form to use depends on the performance criterion we want to minimize or maximize.

Linear Quadratic Optimal control

If the problem at hand should drive the states to a final position, taking into consideration the deviation from the final state only, the cost function would have the following form

$$J(x(t_f)) = x(t_f)^\top H x(t_f). \tag{2.28}$$

This form penalizes the deviation from the final state without considering the input and states trajectory path. The function is quadratic, mainly to deal with the positive and negative values of the deviation without the need to use absolute values in the function. The matrix $H \in \mathbb{R}^{n\times n}$ is a weighing matrix, that is required to be a positive semi-definite matrix.

In many cases, OCPs are solved in such a way to drive the states from $x(t_0)$ to $x(t_f)$, while minimizing the input of the system, a general form of the cost function can either be an absolute value of the input or a quadratic function. The cost function is an integral along the whole path of the input signal

$$J(u(t)) = \int_{t_0}^{t_f} u(t)^\top R u(t)\,\mathrm{d}t, \tag{2.29}$$

where $R \in \mathbb{R}^{m\times m}$ is a real symmetric positive definite matrix.

If we would like instead to minimize the deviation of the states from a certain reference across the complete trajectory, then penalizing the state across an integral is noted as

$$J(x(t)) = \int_{t_0}^{t_f} x(t)^\top Q x(t)\,\mathrm{d}t. \tag{2.30}$$

This minimizes the deviation from a reference across the whole trajectory. Matrix $Q \in \mathbb{R}^{n\times n}$ is a symmetric positive semi-definite matrix. The cost function's value is positive or zero, due to the choice of the matrices Q and R. In the previous cost functions, no reference value was set, thus the reference value is set to zero and the solution to such problem would drive the states values to zero. This is known as a regulator. If a reference value other than zero to be used, this is known as a tracking problem. The only difference is that the states of the system are replaced by the error of the system. The error of the system is introduced as the deviation of the actual measurements of the system from the reference value introduced, this is equivalent to

$$e(t) = Cx_{ref}(t) - Cx(t) = y_{ref}(t) - y(t). \tag{2.31}$$

The final form of the cost function is noted as

$$J(x(t), u(t)) = \frac{1}{2} e(t_f)^\top H e(t) + \frac{1}{2} \int\limits_{t_0}^{t_f} e(t)^\top Q e(t) + u(t)^\top R u(t) \, \mathrm{d}t. \tag{2.32}$$

If the system dynamics are linear, then this type of problem is known as a Linear Quadratic problem.

2.4 Constrained Optimization

Mathematical optimization problems are divided into unconstrained and constrained optimization problems. The unconstrained problem does not require any constraints to be fulfilled, for the trajectory to be considered optimum. The general form of such problem is noted as

$$\min_{x} \quad f(x). \tag{2.33}$$

In OCP, there are equality and inequality constraints, that the solution must obey. In [13], several constraint examples are shown. The system dynamics are equality constraints, that must always be considered when formulating the OCP. The system dynamics are noted as

$$\dot{x} = f(x(t), u(t)). \tag{2.34}$$

Different constraints can be applied to an OCP, so that a certain requirement is considered. In most cases, an input signal or state must be bounded between two values. The bounded constraints are applied as

$$\begin{aligned} x_{min} \leq x(t) \leq x_{max} \\ u_{min} \leq u(t) \leq u_{max}. \end{aligned} \tag{2.35}$$

Initial and final states are constraints that are satisfied. Depending on the system and the required outputs, these constraints can be equality or inequality constraints. These constraints are noted as

$$g(t_0, t_f, x(t_0), x(t_f)) \leq 0. \tag{2.36}$$

Path constraints are required, when conditions have to be satisfied along the states trajectory. Path inequalities are noted as

$$h(t, x(t), u(t)) \leq 0. \tag{2.37}$$

For the OCP applied to the heat equation, the system dynamics, initial temperature and the bounded input signals are the constraints considered.

Numerical optimization deals with solving mathematical optimization problems numerically. The first step to solving an optimization problem is to create a mathematical model, which consists of the decision variables, constraints and objective function. This is analogous to an OCP formulation. Based on the model, an appropriate algorithm can be implemented computationally to obtain the optimal trajectory. In this section, basic concepts and definitions are introduced, together with a class of numerical optimization programs, known as quadratic programming. The definitions and examples in this section are according to [15].

The general form of a mathematical optimization problem with equality and inequality constraints is noted as

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ \text{s.t} \quad & c_i(x) = 0, i = 1, \dots, m \\ & c_i(x) \geq 0, i = 1, \dots, q \end{aligned} \tag{2.38}$$

A solution x^* to a constrained problem must satisfy the constraints. This introduces us to the first definition.

Definition 1 ***(Feasible set)***
A feasible set Ω is the set of points x that satisfies the constraints of the problem:

$$\Omega = \{x | c_i(x) = 0, i = 1, \dots, m; c_i(x) \geq 0, i = 1, \dots, q\}. \tag{2.39}$$

An example can be shown in the following problem [10, 11]

$$\begin{aligned} \min_{x \in R^2} \quad & \sqrt{x_2} \\ \text{s.t} \quad & x_2 \geq 0 \\ & x_2 \geq (2x_1)^3 \\ & x_2 \geq (-x_1 + 1)^3 \end{aligned} \tag{2.40}$$

The solution to the previous example can be shown in Figure 2.7.

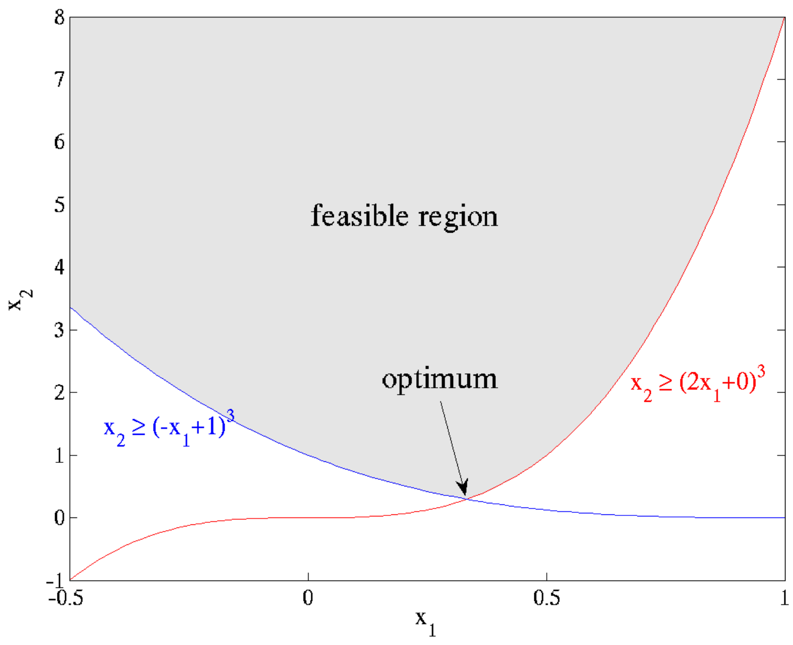


Figure 2.7: solution to a constrained nonlinear problem showing the feasible region of the states [10, 11].

As shown in Figure 2.7, the feasible region lies in the area, where all constraints are satisfied. The optimum value of the objective function, is the value of function, where its argument x_2 is satisfying all the constraints.

Different types of computational programs are introduced based on the formulation of the mathematical optimization problem. If the objective function, together with constraints of the problem are known to be linear, the program used to solve such problem is referred to as a linear program. Another Form is non linear programs (NLP), which many of its algorithms can be applied to solve OCPs. A form of an NLP is a Quadratic program (QP). A General QP is written as

$$\begin{aligned} \min_{x} \quad & \frac{1}{2} x^\top G x + x^\top c \\ \text{s.t} \quad & a_i^\top x = b_i, i = 1, \ldots, m \\ & a_i^\top x \geq b_i, i = 1, \ldots, q \end{aligned} \tag{2.41}$$

where $G \in \mathbb{R}^{n \times n}$ is a symmetric matrix, $c, x, a_i \in \mathbb{R}^n$.

NLP solvers and algorithms usually search for a feasible solution x, which is only local minimum. A local minimum guarantees that the value of the function is less than the value of the function at neighbouring feasible sets x. However, the solution is not globally minimum, which is the least value of the function at any feasible set x. A solution is guranteed to be a global minimum, if the objective function is a convex function. The difference between a global and a local minimum is presented in Figure 2.8.

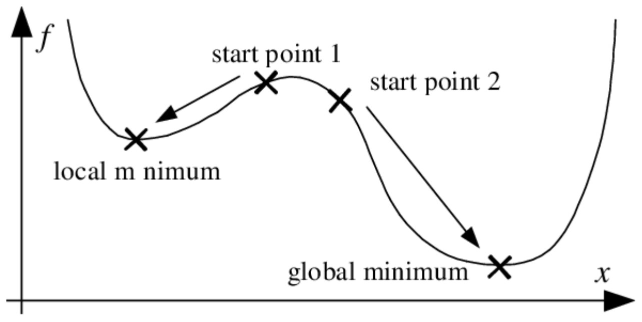


Figure 2.8: An example of a local and a global minimum and illustration of how a solver typically finds a solution [6].

Definition 2 ***(Convex set)*** *A convex set $S \in \mathbb{R}^n$ is said to be convex, if a straight line connecting two points, that lie inside the set, lies completely inside the set.*

$$x, y \in S, \alpha x + (1-\alpha)y \in S, \forall \alpha \in [0,1]$$

Definition 3 ***(Convex function)*** *A function f is said to be convex, if its domain S is a convex set, and for any two pints $x, y \in S$, the following property applies:*

$$f\left(\alpha x + (1-\alpha)y\right) \leq \alpha f(x) + (1-\alpha)f(y), \forall \alpha \in [0,1]$$

The definition of a convex function is visualised in Figure 2.9.

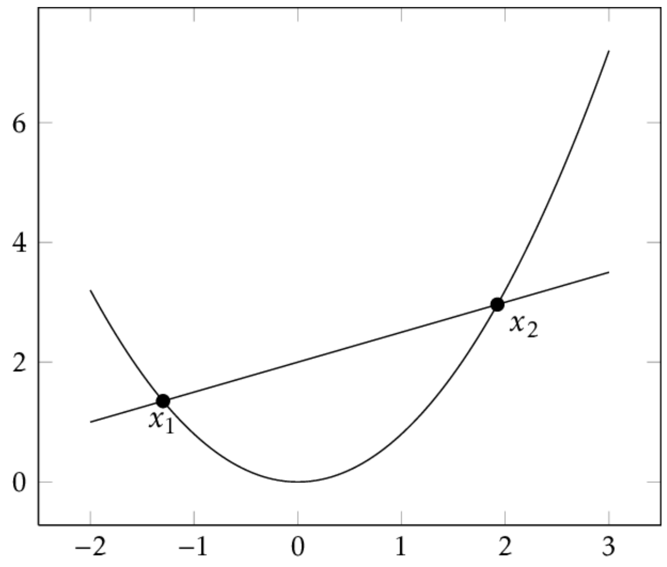


Figure 2.9: A convex function with an illustration to the definition: A straight line connecting two points on the domain of the function, always lies above the curvature of the function [21, p. 14].

A convex program is much easier to solve, as the cost function only has a global minimum. A NLP is said to be convex, if the objective function is convex, equality constraints are linear and inequality constraints are convex.

We are going to implement a LQ-optimal control with bounded input constraints to solve our problem. The objective function is quadratic with positive semi-definite Q matrix and a positive definite R matrix, this means that the objective function is convex. The constraints are linear, which is also convex. This means that the complete problem is convex.

2.5 Direct collocation method

The numerical methods used for trajectory optimization are divided into two categories known as the indirect and direct methods. Direct methods depend on discretizing the continous function and its constraints and reformulating the problem into a NLP that can be solved directly with a suitable algorithm to the problem. These methods are motivated nowadays as they scale well to large OCPs, which is why it is being widely developed in the industry and due to the high computational power and the availability of specific software that solves that kind of problems available [19]. The methods differ from one another based on the form of approximation of the decision variables. The method that discretizes the states and input variables is called direct collocation method [18, 22]. The continous functions in the OCPs are approximated to polynomials [13].

In direct collocation methods, the simulation time is divided into N sampling points with equal sampling periods, that is Δt is equivalent for all values of $k = 1, 2, \cdots, N$. The values of $x(t), u(t)$ are calculated at each sampling point producing the matrices X, U. In this notation, $x[k]$ corresponds to the value of $x(t)$ at $t = t_k$ and $u[k]$ is the value of $u(t)$ at the same time point. The lower and upper bounds of states and control inputs are $U_{min}, U_{max}, X_{min}, X_{max}$.

Euler Forward method

Euler forward method is an explicit method that is used for discretization of differential equations for numerical solutions. Euler forward method depends on the linear approximation of the differential equation based on the calculation of the tangent of the function. A time interval is discretized into $N-1$ points, giving a time vector $T[1 : N-1]$ The system dynamics to our problem can be discretized for $\Delta t = t_{k+1} - t_k$ as

$$x[k+1] - x[k] = \Delta t(Ax[k] + Bu[k]). \tag{2.42}$$

In Figure 2.10, a time step in euler forward method is shown.

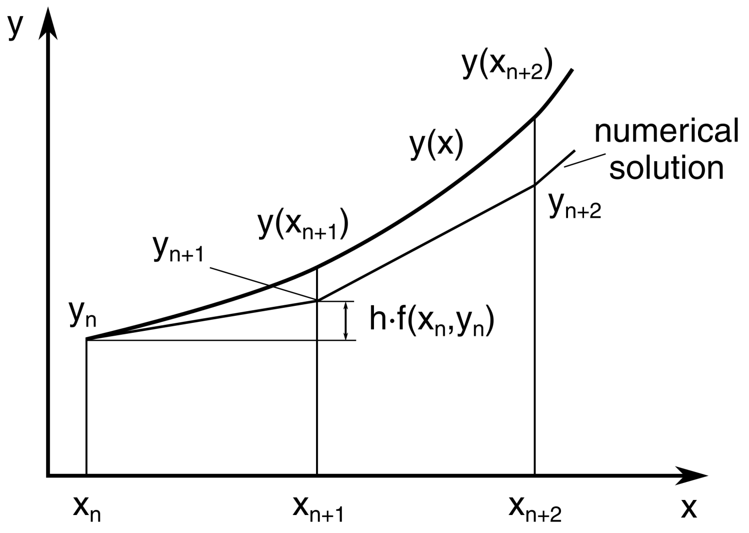


Figure 2.10: Representation of Euler method [9]

Other constraints are discretized as

$$\begin{aligned} X_{min} \leq x[k] \leq X_{max}, \\ U_{min} \leq u[k] \leq U_{max}. \end{aligned} \tag{2.43}$$

The cost function in Equation 2.27 is rewritten as

$$J = \frac{1}{2} J(x[1], T[1], x[N], T[N]) + \frac{\Delta t}{2} \sum_{k=1}^{N-1} \omega(x[k], u[k]). \tag{2.44}$$

2.6 Optimal control of heat equation

The OCP for the boundary control of heat equation is formed using

- the cost function as introduced in Section 2.3.
- Constraints and initial states as introduced in Section 2.4.
- The approximated heat equation as introduced in Section 2.2.

The discretized form of the OCP is introduced in Section 2.5. The OCP is noted as

$$\begin{aligned} \min_{\theta,U} \quad & \frac{\Delta t}{2} \sum_{k=1}^{N-1} e[k]^\top Q e[k] + u[k]^\top R u[k] \\ \text{s.t} \quad & \theta[k+1] = \theta[k] + \Delta t (A\theta[k] + Bu[k]) \\ & \theta[1] = \theta_{init} \\ & u_{min} \leq u[k] \leq u_{max} \end{aligned} \tag{2.45}$$

$\theta[k]$ is the temperature vector at time step k, where the temperature at each discretized node is considered a state. $e[k]$ is the error between the output and the reference value as in Equation (2.31) at time step $k \in \{1, \cdots, N\}$.

The two dimensional problem has the same structure, with $u[k] \in \mathbb{R}^m$ for multiple input signals.

The weighing matrices are chosen as

$$\begin{aligned} Q &= \gamma \; C^\top \; C, \\ R &= \phi \; I, \end{aligned}$$

where $\gamma \geq 0$ and $\phi > 0$, output matrix $C \in \mathbb{R}^{q \times n}$ and identity matrix $I \in \mathbb{R}^{m \times m}$.

Chapter 3

Software

After the formulation of the NLP, a mathematical optimization framework is used to solve such problems. Julia is a dynamic programming language, created for scientific and computing applications, while keeping the performance criterion better than other static programming languages. A brief introduction can be found in [5]. In Julia, there are packages introduced for mathematical optimization problems, such as JuMP [7] and NLOptControl [8]. In this chapter, an overview of JuMP and NLOptControl using the mechanical oscillator example introduced in Section 2.1 is provided. Both frameworks are briefly compared to unveil their benefits and limitations.

3.1 JuMP.jl

JuMP [3, 7] is a mathematical modelling language embedded in Julia. The package has the form of a high level programming language. JuMP is built on Julia, which allows the usage of all Julia packages in the built model. In [7], benchmarks are used to compare JuMP to other commercial modelling tools, showing JuMP's efficiency in solving optimization problems.

JuMP is designed to take advantage of Julia's syntactic macros [7], providing suitable syntax for modelling without the usage of parsers and overcoming the problem of operator overloading, which is used in other programming languages. JuMP does not solve the optimization problems itself. It supports various commercial and open source solvers for different types of optimization problems. IPOPT [24] is used for the optimal control problems introduced, because it is open source and designed for sparse large scale NLPs. It uses an interior point line search algorithm to find the minimum value of the objective function. It is important to note that IPOPT is a local solver, meaning that if the objective function is not convex, there is no gurantee the solution found is the global minimum.

The first step is to include the packages that are used and to initialize a model.

```julia
using JuMP,IPopt
#model initialization
mod = Model( optimizer_with_attributes( Ipopt.Optimizer ,  "max_iter" => 1000))
```

The system parameters are initialised. The simulation final time is $t_f = 10[s]$, with time steps $N = 1000$. The sampling time is $\Delta t = \frac{t_f}{N}$. The variables u_{min} and u_{max} are the lower and upper bounds to the input signal.

```julia
ω₀ = 1.0; # Frequency
d =  -20.5; # Damping
A = [0 1; -ω₀ -2*d*ω₀]; # System matrix
B = [0 ; 1]; # Input matrix
N = 1000;           # Time steps
Tf = 10.0;          # Final time
ΔT = Tf/N;          # Sampling time
x10 = 2.0;          #initial value of state $x_1$
x20 = 1.0;          #initial value of state $x_2$
Q = [10 0; 0 100];  #weighing matrix of states
R = 0.01;           #weighing value of input u(t)
```

The input $u(t)$ has upper and lower boundaries. The decision variables are stored using `@variable`.

```julia
@variable(mod, umin <= u[1:N-1] <= umax) #Control signal
@variable(mod, x1[1:N])         # State x₁
@variable(mod, x2[1:N])         # State x₂
```

The `@constraint` and `@NLconstraint` are used to define the constraints of the problem. The system dynamics are discretized as introduced in Section 2.5.

```julia
# Initial values of states
@constraint(mod, x1[1] == x10 )#Initial value of state $x_1$
@constraint(mod, x2[1] == x20 )#Initial value of state $x_2$
# System dynamics.
for k in 1: N-1
@constraint(mod, x1[k+1] == x1[k]  + 0.5 * ΔT *(A[1,:]' * [x1[k],x2[k]] + B[1,1] *
u[k]))
@constraint(mod, x2[k+1] == x2[k]  + 0.5 * ΔT *(A[2,:]' * [x1[k],x2[k]] + B[2,1] *
u[k]))
end
```

The objective function is initialized using `@NLobjective` macro. The objective function is quadratic and is in trapezoidal form. The model is passed to the `optimize!()` function to start solving the problem.

```
@NLobjective( mod, Min, 0.5 * ΔT * sum(  Q[1,1]*(x1[k])^2 + Q[2,2]*(x2[k])^2
+ R*(u[k])^2 for k=1:N-1 ))
optimize!(mod)
```

The results of the simulation can be plotted using any of the available packages. The results are simulated using the Plots package and are shown in Figures 3.1 and 3.2.

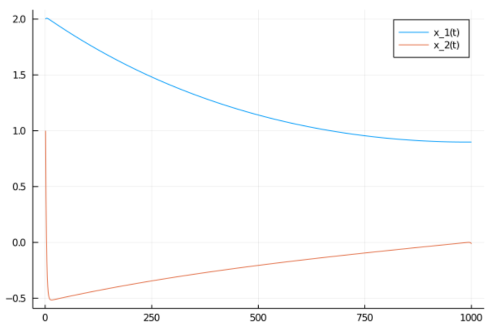


Figure 3.1: The states trajectory

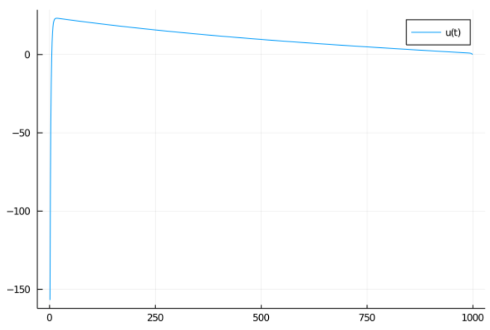


Figure 3.2: Input signal trajectory

3.2 NLOptControl

NLOptControl is a package built on JuMP to solve OCPs. It is designed so that the modelling of the problem can be written in Bolza form as described in Section 2.3. It has different collocation schemes already implemented. Examples on how to use could be found in [8].

The model is initialised with a function called `define()`, where the number of states, inputs, lower and upper bounds and initial and final states are passed as arguments of the function.

```
#model definition
n=define(numStates = 2, numControls = 1, X0 = [2.0, 1.0],
CL = [-1000.0], CU = [1000.0]);
```

Names can be given to the states and input variables, where each name is a symbol.

```
#model definition
states!(n,[:x,:y],descriptions=["x(t)","y(t)"]);
controls!(n,[:u],descriptions=["u(t)"]);
```

The system dynamics are passed as an array of symbols to the function `dynamics!()`. The variable j has to be used in the brackets, which is replaced by the time step variables.

```
#System's dynamics
dx=[:(y[j]),:((-1.0 * x[j]) + (41 * y[j]) + u[j])];
dynamics!(n,dx);
```

The configuration of the OCP is set by the method `cofigure!()`. The method arguments are the number of time steps N, the final time tf and the integration scheme.

```
#Problem's configuration
configure!(n,N = 1000; (:integrationScheme=>:trapezoidal),(:tf=>10.0))
```

The objective function is initialised. If the objective function contains an integration operation, an `integrate!()` method is called. NLOptControl uses JuMP macros to define the objective function and uses the `optimize!()` method for the solution of the problem.

```
#objective function and optimization
obj = integrate!(n,:(((10 * x[j]^2 + 100 * y[j]^2))+ (0.01 * u[j]^2)));
@NLobjective(n.ocp.mdl, Min, obj);
optimize!(n)
```

The results are simulated using the Plots package and are shown in Figures 3.3 and 3.4.

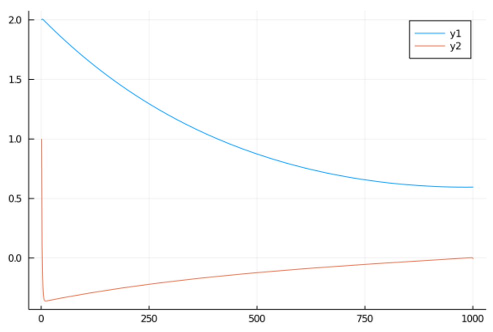


Figure 3.3: The states trajectory

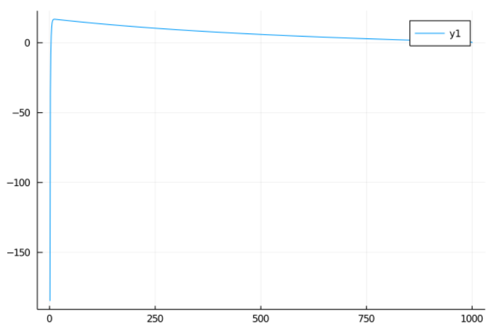


Figure 3.4: Input signal trajectory

In Chapter 4, we will focus on JuMP because the recent state of NLOpt-Control does not seem to be able to work in our context with large-scale problems.

Chapter 4

Implementation and Results

In the current chapter, the implementation of the OCP for the heat equation is presented. In Section 4.1, the simulation paramers of the one dimensional heat equation are presented, followed by the simulation results of the OCP for various cases. In Section 4.2, the simulation parameters of the two dimensional heat equation are presented. The simulation results in Section 4.2 show the results for heating one plate using multiple input signals and heating multiple plates.

4.1 The one Dimensional heat equation

The approximation of the heat equation is introduced in Section 2.2. The characteristics of the rod are noted in Table 4.1.

Parameter	Symbol	Value	Unit
Length	L	0.1	m
Discretization points	N	11	
Spatial discretization	Δx	0.01	m
Thermal Conductivity	λ	45	$W/(m \cdot K)$
specific heat capacity	c	460	$J/(Kg \cdot K)$
Material's density	ρ	7800	$Kg \cdot m^3$

Table 4.1: Simulation parameters for one dimensional case.

In further calculations we use $\alpha = \frac{\lambda}{c\rho}$. According to the Table 4.1, the states vector is $\theta \in \mathbb{R}^{11}$, the system matrix $A \in \mathbb{R}^{11\times 11}$. The input source is placed at the left hand side of the rod. The input matrix $B \in \mathbb{R}^{11\times 1}$

The sensor is assumed at $x = L$, which corresponds with the last node of the rod. Thus, the output matrix $C \in \mathbb{R}^{1\times 11}$ is noted as

$$C = \begin{bmatrix} 0 & \cdots & 1 \end{bmatrix}. \tag{4.1}$$

For the numerical stability of the approximated heat equation discussed in Section 2.2, the sampling period has to satisfy the conditional Equation (2.19).

The time step size in simulation is $\Delta t = 10^{-1}$. The number of constraints depend on the time step size, where a smaller step means more constraints to be initialised. The number of calculation steps is noted as

$$k = \frac{t_f}{\Delta t}, \tag{4.2}$$

with $t_f > 0$ as the final simulation time. The system matrix A is a sparse matrix, where nearly all it's entries other than the tridiagonal representing the approximations are zero. In Julia this is done by using the package SparseArrays.

```julia
# Creating the approximation of the heat equation
using SparseArrays
A = spdiagm(-1 => ones( N-1), 0 => -2*ones(N), 1 => ones(N-1));
A[2,1] = 2.0;
A[end - 1,end] = 2.0;
A = (λ/(c * ρ *  dx^2)) * A;
```

Vectors B and C are defined similarly.

The next step is the initialization of the decision variables. These are the temperatures at the $N = 11$ grid points, the input variable and the output variable for k time steps.

```julia
# Creating the approximation of the heat equation
@variable(mod,θ[1:N,1:step]);        #temperature at dx[i = 1:N]
@variable(mod,u[k = 1:step - 1]);    #Input u
@variable(mod, y[1:step]);           #Output y
```

In the following step, we introduce the system dynamics and initial conditions of our states. These are the constraints of our system, as discussed in Section 2.4, which are linear constraints.

```
#Initial states and output of the system
@constraint(mod, θ[:,1] .== θinit)
@constraint(mod, y[1] .== θinit)
#system's dynamics
for k in 1:step - 1,i in 1:N
    @constraint(mod,θ[i,k + 1] .==  θ[i,k] + dt *
                ((A[:,i]'*θ[:,k]) + B[i] * u[k]))
end
#Output of the problem
for k in 1:step - 1
    @constraint(mod,y[k + 1] .==  C' * θ[:,k + 1])
end
```

The variables $\theta init$ is the inital temperature, set to 273 Kelvin.

The cost function is quadratic, therefore a macro for non linear objective function is used.

```
#Cost functional for LQ-OCP
J = @NLexpression(mod,0.5 * dt * sum(Q * (θref - y[k]) ^ 2
                +R * u[k] ^ 2 for k in 1:step-1))
```

The values of Q and R were determined with trial runs. For $R \geq 1$ and $Q = 1$, the temperature across the rod would not increase considerably, as the input signal is penalized with a big value, giving it a small value and quickly driving the input trajectory to zero. Q and R are scalars, where the error consists of one measurement and there is only one input source. This means that the states path needs to be penalized more than the input signal. However, if Q is set too large then this could lead to an overshoot in the temperature across the rod. The time interval differs according to the values of Q and R. As the final time increases, the problem size increases, where the number of constraints is approximately $(N + 1) \times k$. The system was found to reach the wanted reference temperature of 500 Kelvin for the simulation time and the weighing matrices shown in Table 4.2.

Parameter	value
t_f	$3000[s]$
Δt	$10^{-1}[s]$
Q	$10^3 C^\top C$
R	10^{-3}

Table 4.2: Optimization and simulation time parameters.

Three cases are tested to simulate the behaviour of the controller with different weighing values of Q and R. The input signal is used with no upper or lower bounds. In the first case, the values $Q = 1$ and $R = 1$ are used, where the simulation results are shown in Figure 4.1.

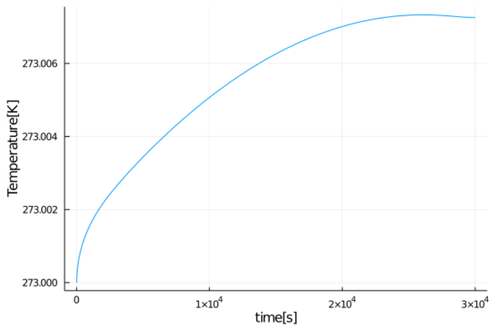


Figure 4.1: Temperature at input node for rod with Q = 1,R = 1

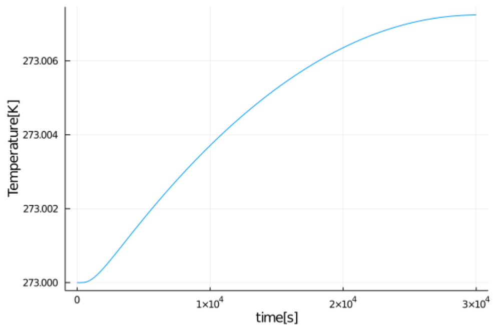


Figure 4.2: Output measured at right side of rod for Q=1 ,R = 1

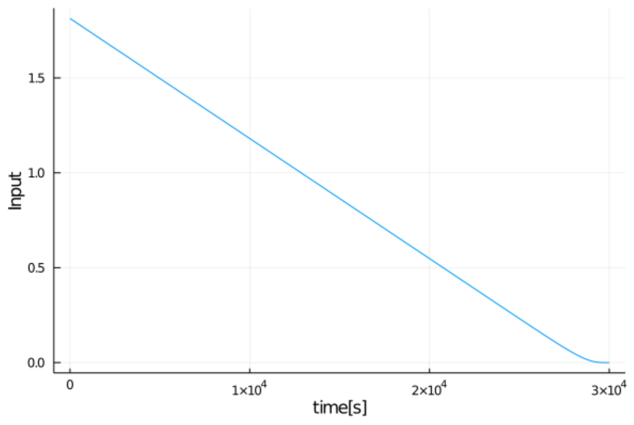


Figure 4.3: Input signal trajectory for Q=1 ,R = 1

From Figure 4.1, it is observed that the rising of the temperature across the rod is too slow and the input signal has a very small value. In the second case, the values chosen are $Q = 10^8$ and $R = 0.1$ to reduce the rising time of the temperature and increase the amount of heat applied on the rod, with no upper or lower bounds to the input signal. The simulation of the temperature distribution across the rod is presented in Figures 4.4 and 4.5 and the input signal trajectory is presented in Figure 4.6.

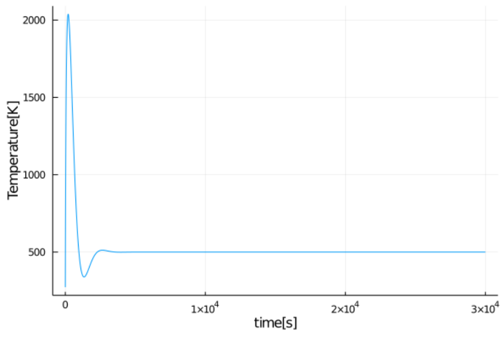


Figure 4.4: The temperature change at the left side of rod $Q = 10^8$,$R = 0.1$.

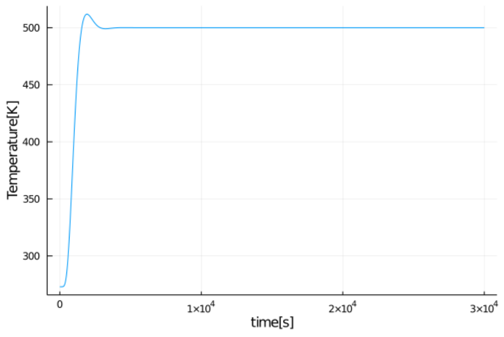


Figure 4.5: Output measured for $Q = 10^8$,$R = 0.1$.

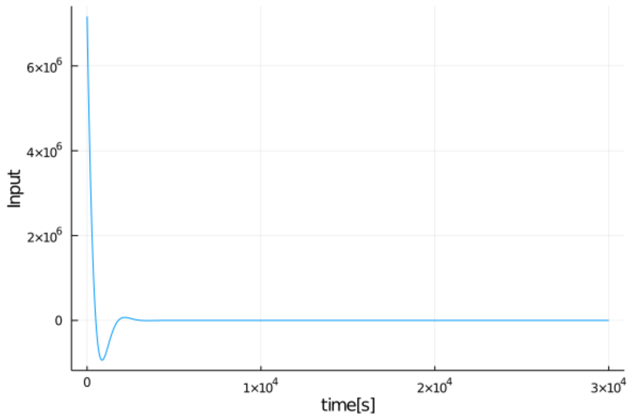
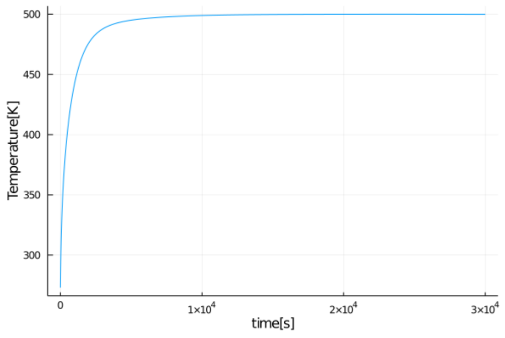


Figure 4.6: Input signal for $Q = 10^8$,$R = 0.1$.

The simulations show the overshoot of the temperature at $x = 0$ and at the measured output. The input source cools down to negative values, which is infeasible. In the third case, the values $Q = 10^3, R = 10^{-3}$ are tested, with no bounds on the input signal. The temperatures at the input nodes and at measured output are shown in Figures 4.7 and 4.8. The input signal trajectory is shown in Figure 4.9.



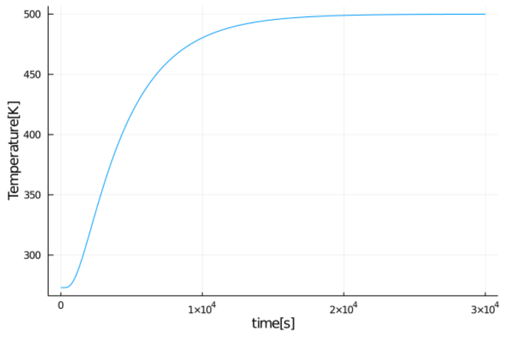


Figure 4.7: The temperature at the left side of rod $Q = 10^3$,$R = 10^{-3}$.

Figure 4.8: Output measured for $Q = 10^3$,$R = 10^{-3}$.

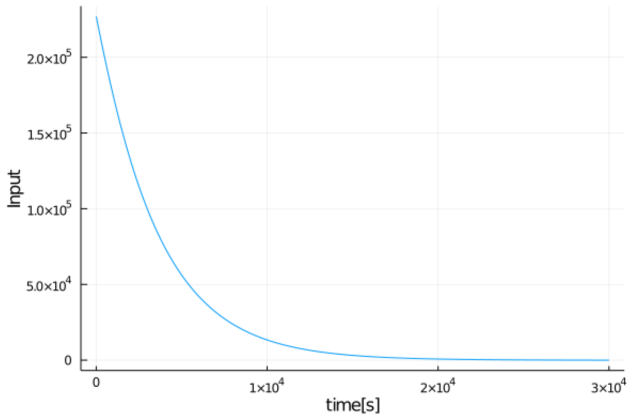


Figure 4.9: Input signal for $Q = 10^3, R = 10^{-3}$.

The weighing values Q and R are determined. The input signals are applied with upper and lower bounds $0 \leq u(t) \leq 15 \cdot 10^3$.

```
#Bounded input signal: u_max = 15\cdot 10^3, u_min = 0.0
@variable(mod,u_min <= u[k = 1:step - 1] <= u_max);    #Input u
```

The simulation time is increased $t_f = 10000$. The temperature trajectory at the input node and the measured output are shown in Figures 4.10 and 4.11. The input signal trajectory is shown in Figure 4.12.

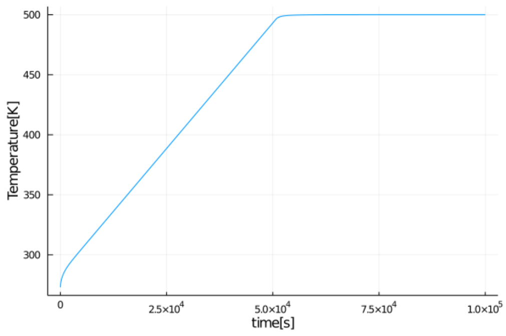


Figure 4.10: The temperature change at the left side of rod for $Q = 10^3$,$R = 10^{-3}$ with $0 \leq u(t) \leq 15 \cdot 10^3$.

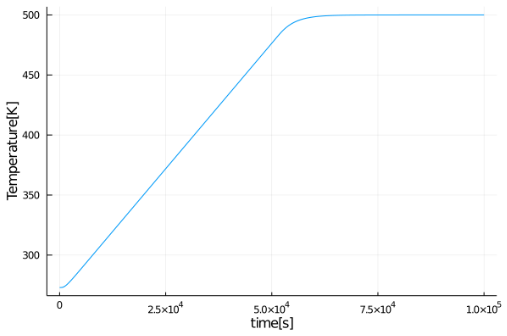


Figure 4.11: Output measured for $Q = 10^3$,$R = 10^{-3}$ with $0 \leq u(t) \leq 15 \cdot 10^3$.

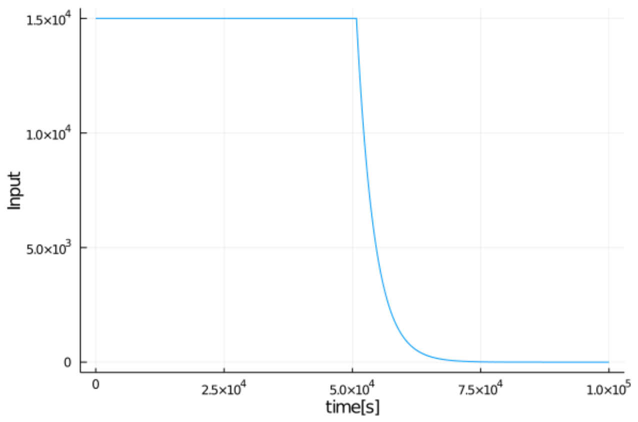


Figure 4.12: Input signal for $Q = 10^3$,$R = 10^{-3}$ with $0 \leq u(t) \leq 15 \cdot 10^3$.

4.2 Two Dimensional heat equation

The two Dimensional heat equation is experimented on a plate, with the simulation parameters presented in Table 4.3.

Parameter	Symbol	Value	Unit
Length	$L_x = L_y$	0.1	m
Discretization points	$N_x = N_y$	11	
Spatial discretization	$\Delta x = \Delta y = h$	0.01	m
Thermal Conductivity	λ	45	$W/(m \cdot K)$
specific heat capacity	c	460	$J/(Kg \cdot K)$
Material's density	ρ	7800	$Kg \cdot m^3$

Table 4.3: Simulation parameters for two dimensional case.

The system matrix $A \in \mathbb{R}^{121 \times 121}$ as described in section 2.2, The input matrix contains multiple input entries at the first, sixth and eleventh node of the matrix, where$B \in \mathbb{R}^{121 \times 3}$. This is equivalent to having three input sources at different discretized nodes of the bottom side of the plate being tested. Three output readings are taken from the upper side of the plate, at the first, sixth and last node on the upper side. The output matrix is $C \in \mathbb{R}^{1 \times 121}$. The chosen time and weighing parameters are noted in Table 4.4.

Parameter	value
t_f	$6000[s]$
Δt	$1.5[s]$
Q	$10^4 C^\top C$
R	10^{-3}

Table 4.4: Optimization and simulation time parameters for two dimensional case.

In the following section, the temperatures at the bottom of the plate (input boundary) correspond to the positions $(0, 0)$, $(L/2, 0)$, $(L, 0)$ and the temperatures at the top of the plate(output boundary) correspond to $(0, L)$, $(L/2, L)$, (L, L). The temperature trajectories at these position are presented in Figures 4.13 and 4.14. The input signals are presented in Figure 4.15.

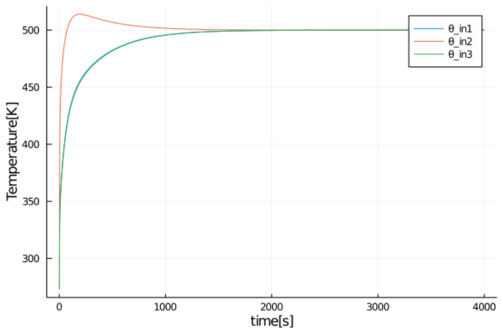


Figure 4.13: Trajectories of the temperature input nodes.

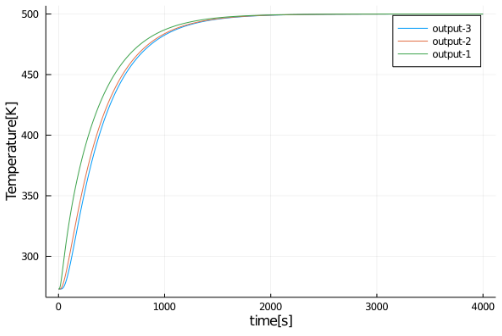


Figure 4.14: Trajectories of the measured temperatures.

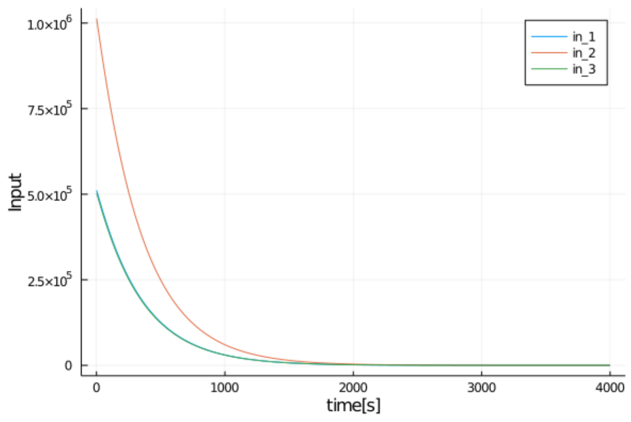


Figure 4.15: Input signal of each of the heat sources.

For achieving bounded input signals, we apply a constraint between $u_{min} = 0$ to $u_{max} = 15000$ to all three input sources. The varied controller parameters are noted in Table 4.5.

Parameter	value
t_f	$40000[s]$
Δt	$1.5[s]$
Q	$10^4 C^\top C$
R	10^{-4}

Table 4.5: Optimization and simulation time parameters for two dimensional case, with bounded input signals.

The simulations of the temperature trajectories at the bottom of the plate (input boundary) correspond to the positions $(0,0), (L/2,0), (L,0)$ and at the top of the plate (output boundary) correspond to $(0,L), (L/2,L), (L,L)$ are presented in Figures 4.16 and 4.17. The input signals are shown in Figure 4.18.

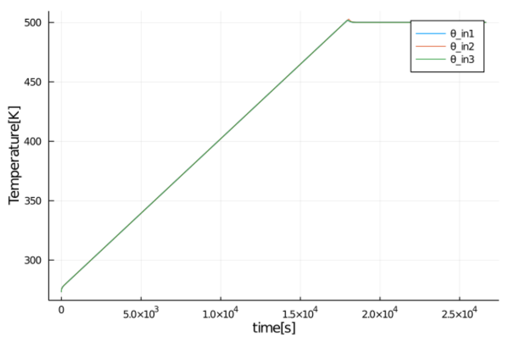


Figure 4.16: Trajectories of the temperature at input nodes.

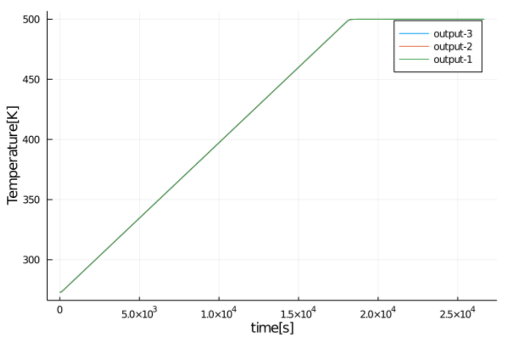


Figure 4.17: Trajectories of the measured temperatures.

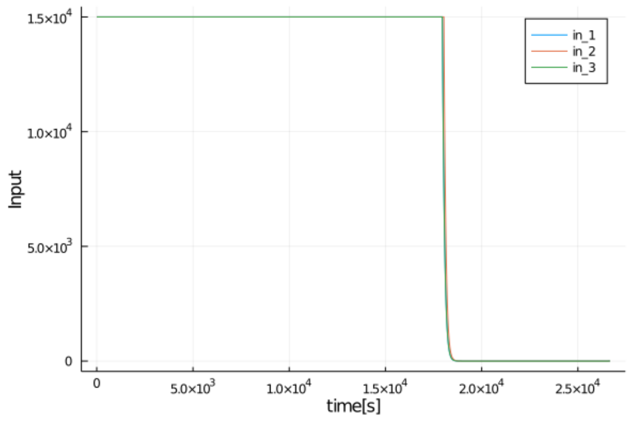


Figure 4.18: bounded Input signal of each of the heat sources.

The final case is heating several two dimensional bodies with various heating sources. Three two dimensional bodies are heated using a heating source for each body. The simulation parameters are shown in Table 4.6.

Parameter	Symbol	Value	Unit
Width	L_x	0.03	m
Length	L_y	0.1	m
Discretization points (Width)	N_x	4	
Discretization points (Length)	N_y	12	
Spatial discretization	$\Delta x = \Delta y = h$	0.01	m
Thermal Conductivity	λ	45	$W/(m \cdot K)$
specific heat capacity	c	460	$J/(Kg \cdot K)$
Material's density	ρ	7800	$Kg \cdot m^3$

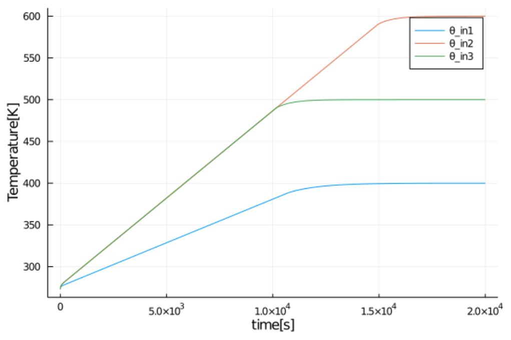
Table 4.6: Simulation parameters for two dimensional case of heating several bodies.

The system matrix $A \in \mathbb{R}^{48\times 48}$ as described in section 2.2. The input matrix contains multiple input entries placed at the lower side of the bodies, where$B \in \mathbb{R}^{48\times 3}$. Two output readings are taken from the upper side of each body, with $C \in \mathbb{R}^{1\times 48}$. Three state vectors are used, $\theta_i \in \mathbb{R}^{48}, i \in 1, 2, 3$. The reference temperatures are set differently for each body, where the first is heated to 400 Kelvin, the second body is heated to 600 Kelvin and the third body is heated to 500 Kelvin. The input signals are $0 \leq u(t) \leq 15 \cdot 10^3$. The controller and simulation parameters are noted in Table 4.7.

Parameter	value
t_f	$30000[s]$
Δt	$1.5[s]$
Q	$10^4 C^\top C$
R	10^{-4}

Table 4.7: Optimization and simulation time parameters for two dimensional case of heating several bodies, with bounded input signals.

The simulations of the temperature trajectories at the bottom of the plate (input boundary) correspond to the positions $(0, 0)$ for the first plate, $(L/2, 0)$ for the second plate and $(L, 0)$ for the third plate are presented in Figure 4.19. The simulation trajectories at the top of the plate (output boundary) correspond to $(L_x, L_y), (L_x, L_y), (L_x, L_y)$ are presented in Figure 4.20. The input signals are shown in Figure 4.21.



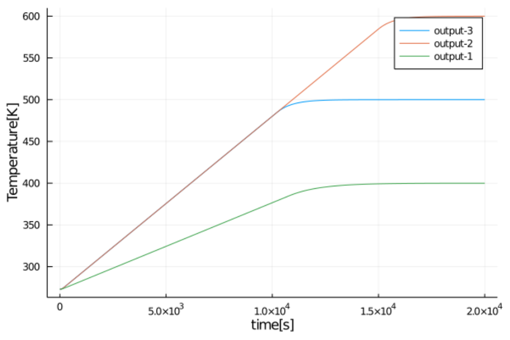


Figure 4.19: The temperature at the input nodes, with $Q = 10^4$,$R = 10^{-4}$ with $0 \leq u(t) \leq 15 \cdot 10^3$.

Figure 4.20: The measured temperature, with $Q = 10^4$,$R = 10^{-4}$ with $0 \leq u(t) \leq 15 \cdot 10^3$.

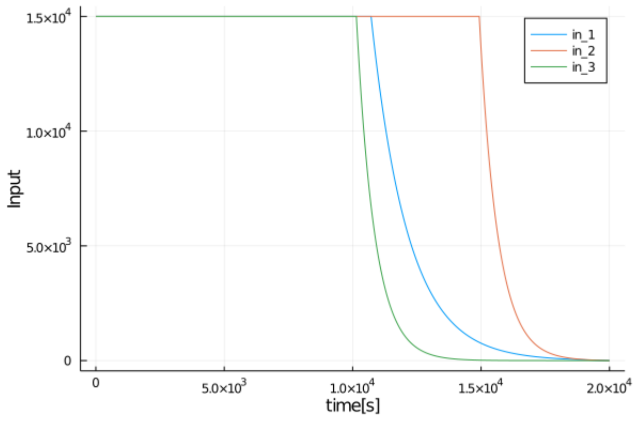


Figure 4.21: The input signals, with $Q = 10^4$,$R = 10^{-4}$ with $0 \leq u(t) \leq 15 \cdot 10^3$.

Chapter 5

Conclusion

In this work, we took interest in the optimal control problem of the heat equation with boundary input and additional constraints on the input value. A stable numerical approximation of the one dimensional and two dimensional heat equation are presented using finite difference method. An optimal control problem is formulated and a solution using direct collocation method is discussed.

A short introduction to JuMP and NLOptControl are presented, which are suitable packages to solve the derived problem. A JuMP model is implemented to simulate the results of a LQ-optimal control applied.

The work introduced in this thesis can be generalized to systems described with parabolic PDEs. Depending on the requirements of the research, A finite element scheme could be also used, if the accuracy of the FDM is not sufficient.

A model predictive controller is an iterative process of solving a constrained optimal control problem for a short time interval $[t, t+T]$ and only implementing the first control input signal. Our work can be used as the base of the structure of a model predictive controller. This is done by reducing the integral period to a very short period and iterating, while only applying the $u[1]$ and reinitialising the problem, where the current states of the system are used as the initial values of the problem.

Bibliography

[1] Joel AE Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[2] Sergey Andreev. System of energy-saving optimal control of metal heating process in heat treatment furnaces of rolling mills. *Machines*, 7(3):60, 2019.

[3] Ing Eckhard Arnold. Jump-kurzbeschreibung. 2018.

[4] Christoph Josef Backi and Jan Tommy Gravdahl. Optimal boundary control for the heat equation with application to freezing with phase change. In *2013 Australian Control Conference*, pages 409–414. IEEE, 2013.

[5] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[6] Juan A Carretero and Meyer A Nahon. A genetic algorithm for calculating minimum distance between convex and concave bodies. pages 18–22, 2001.

[7] Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[8] Huckleberry Febbo, Paramsothy Jayakumar, Jeffrey L Stein, and Tulga Ersal. Nloptcontrol: A modeling language for solving optimal control problems. *arXiv preprint arXiv:2003.00142*, 2020.

[9] Steven G. Johnson. Euler's method explained with examples. `https://www.freecodecamp.org/news/eulers-method-explained-with-examples/`.

[10] Steven G. Johnson. The nlopt nonlinear-optimization package. `http://github.com/stevengj/nlopt`.

[11] Steven G. Johnson. The nlopt nonlinear-optimization package. `https://nlopt.readthedocs.io/en/latest/NLopt_Tutorial/`.

[12] Rudolf E Kalman. On the general theory of control systems. In *Proceedings First International Conference on Automatic Control, Moscow, USSR*, pages 481–492, 1960.

[13] Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

[14] Donald E Kirk. *Optimal control theory: an introduction.* Courier Corporation, 2004.

[15] Jorge Nocedal and Stephen Wright. *Numerical optimization.* Springer Science & Business Media, 2006.

[16] Geert Jan Olsder and Jacob Willen van der Woude. *Mathematical systems theory*, volume 4. VSSD Delft, 2005.

[17] Peter J Olver. *Introduction to partial differential equations.* Springer, 2014.

[18] Anil V Rao. Trajectory optimization: a survey. In *Optimization and optimal control in automotive systems*, pages 3–21. Springer, 2014.

[19] Helena Sofia Rodrigues, M Teresa T Monteiro, and Delfim FM Torres. Optimal control and numerical software: an overview. *arXiv preprint arXiv:1401.7279*, 2014.

[20] Stephan Scholz. Mechanical system with spring, mass and damping. `https://nbviewer.jupyter.org/github/stephans3/control-engineering-edu/blob/master/2020-1-Summer/text/jupyter/CE_2020_04_modeling_mechanical_oscillator.ipynb`.

[21] Daniel Simon. *Fighter Aircraft Maneuver Limiting Using MPC: Theory and Application*, volume 1881. Linköping University Electronic Press, 2017.

[22] Russ Tedrake. Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832). *Downloaded in Fall*, 2014.

[23] Tilman Utz, Sönke Rhein, and Knut Graichen. Transformation approach to constraint handling in optimal control of the heat equation. *IFAC Proceedings Volumes*, 47(3):9135–9140, 2014.

[24] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.