Lukas Kasper

MODELING OF THE PHASE CHANGE MATERIAL OF A HYBRID STORAGE USING THE FINITE ELEMENT METHOD

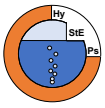
TU WIEN Academic Press

Forschungsgeleitete Lehre

Lukas Kasper

MODELING OF THE PHASE CHANGE MATERIAL OF A HYBRID STORAGE USING THE FINITE ELEMENT METHOD

This publication emerged from the project HyStEPs, which is funded by the Austrian Research Promotion Agency (FFG) with grant number 868842. It was supported by the Institute for Energy Systems and Thermodynamics (IET) of the Faculty of Mechanical and Industrial Engineering, TU Wien.



Lukas Kasper

MODELING OF THE PHASE CHANGE MATERIAL OF A HYBRID STORAGE USING THE FINITE ELEMENT METHOD

TU WIEN Academic Press

Cite as:
Kasper, L. (2020). *Modeling of the Phase Change Material of a Hybrid Storage using the Finite Element Method.* TU Wien Academic Press. https://doi.org/10.34727/2020/isbn.978-3-85448-037-2

TU Wien Academic Press 2020

c/o TU Wien Bibliothek
TU Wien
Resselgasse 4, 1040 Wien
academicpress@tuwien.ac.at
www.tuwien.at/academicpress

ISBN (print): 978-3-85448-036-5
ISBN (online): 978-3-85448-037-2

Available online: https://doi.org/10.34727/2020/isbn.978-3-85448-037-2

Media proprietor: TU Wien, Karlsplatz 13, 1040 Wien
Publisher: TU Wien Academic Press
Author (responsible for the content): Lukas Kasper
Production: Facultas Verlags- und Buchhandels AG

Proofreading: Jennifer Weck

This publication is a revision of the thesis *Modeling of the Phase Change Material of a Hybrid Storage using the Finite Element Method*, written by Lukas Kasper at TU Wien on June 12, 2019. This thesis was published at TU Wien: https://permalink.catalogplus.tuwien.at/AC15369397.
It was written in the framework of the Master programme *Physical Energy and Measurement Engineering* at the Department of Physics of TU Wien. The thesis was supervised and graded by Univ.-Prof. Dr. René Hofmann (supervisor and reviewer), Dr. Alexander Schirrer (co-supervisor) and Dr. Martin Koller.

Preface

When Lukas Kasper reached out to me and my colleague Priv.-Doz. Dr. techn. Alexander Schirrer in 2018, looking for an interesting and application-oriented topic for his diploma thesis, we seized the opportunity to include him in our recently started research project, Hybrid Storage for Efficient Processes [HyStEPs]. The aim of HyStEPs is to develop an all-in-one storage concept to increase the efficiency and flexibility of industrial processes. This project provides interesting research questions and work items that could be answered in the course of a thesis under qualified supervision. The following thesis is the result of the successful symbiosis in the course of this project between an engaged student and experienced scientists. At the TU Wien, great emphasis is put on the participation of students in current research activities. At the Institute for Energy Systems and Thermodynamics, we try to involve students in our research groups as early as possible.

Experience has shown that the concept of research-oriented teaching provides considerable advantages for students, researchers and teachers. Students can benefit from research engagement through increased motivation, a great work environment and direct interchange with colleagues with professional expertise. Students have the unique opportunity to obtain the latest findings from research in teaching and can gain valuable insights into ongoing work. On the one hand, this can lead to a broader range of methodical results; on the other hand, it can also encourage application in a wider variety of subject areas. Methodically, there is the option to deepen theoretical or analytical investigations in related fields. Finally, diploma theses are a viable, compact format for collaboration with industrial partners. These aspects not only enable fruitful research work - they also raise the quality of teaching, with the goal to develop scientific excellence and impart comprehensive competence. This diploma thesis serves as an excellent example in that regard.

Lukas Kasper is currently working on his doctoral thesis, wherein he continues his research in the area of modelling and optimization of industrial energy systems.

Univ.Prof. Dr. techn. René Hofmann

January, 2020

Acknowledgements

I am very grateful to my thesis advisor, Prof. René Hofmann, who gave me the chance to engage myself in this research activity. It was a great experience to be involved in this challenging topic. I wish to thank him for his invaluable advice, which has been essential for this work.

Thank you to all my colleagues from the Institute for Energy Systems and Thermodynamics for the fruitful teamwork and for involving me in the HyStEPs project. I am particularly grateful for the assistance given by Dr. Martin Koller, who also helped me proofread this thesis.

Furthermore, I want to acknowledge my colleagues from the Control and Process Automation Research Unit, especially for conducting numerous simulations on their spontaneously arranged computing grid.

Special thanks to Dr. Alexander Schirrer for lending his expertise in MATLAB and computer science in general.

Finally, I would like to express my gratitude to my parents and my girlfriend Nicki for their constant support throughout my studies. Thank you for your continuous encouragement and for your unfailing patience.

Abstract

The progressive decarbonisation process and the increased share of renewable energy sources in the grid has increased the need for the development of new approaches to store energy and to increase the efficiency of industrial processes. Increasingly, latent heat thermal energy storage systems are being used, which exploit phase change materials to store a large amount of energy during the phase change. A novel approach to increasing the efficiency of the commonly used Ruths steam storage is currently being investigated in the project HyStEPs, a project funded by the Austrian Research Promotion Agency (FFG) with grant number 868842. In this concept, a container filled with phase change material is placed at the shell surface of the Ruths steam storage.

In this diploma thesis, and in contribution to the HyStEPs project, the phase change material of this hybrid storage is modelled in two dimensions using the finite element method. The apparent heat capacity method is applied in a MATLAB implementation and considers heat transfer by both conduction and natural convection. Furthermore, the developed code can handle any desired layout of materials arranged on a rectangular domain. The model was successfully validated using an analytical solution and experimental data, and a cross-validation showed corroboration with the results of the CFD software ANSYS Fluent. A parameter study was conducted and the behaviour of different dimensions and orientations of the phase change material cavity was investigated. The effect of natural convection was found to cause significantly varying behaviour of the studied cavities with different orientation during the charging process, while the effect was found to be negligible during the discharging process.

Table of Contents

Nomenclature

Variables and Parameters

α	heat transfer coefficient $\left[W/(m^2K)\right]$
α_L	thermal diffusivity $\left[m^2/s\right]$
β	linear thermal expansion coefficient $[1/K]$
Δl_m	specific latent heat $[J/kg]$
Δt	simulation time step size $[s]$
Δx, Δy	mesh resolution in x and y direction $[m]$
ϵ	mushy region temperature range $[K]$
λ_{St}	dimensionless parameter occuring in the solution of the Stefan problem
$\mathbf{f}$	force density vector $\left[N/m^3\right]$
$\mathbf{g}$	gravitational acceleration vector $\left[m/s^2\right]$
$\mathbf{u}$	velocity vector $[m/s]$
μ	dynamic viscosity $\left[(Ns)/m^2\right]$
ρ	density $\left[kg/m^3\right]$
ρ_0	reference density $\left[kg/m^3\right]$
ξ, η	transformed space coordinates $[m]$
a_m	fraction of total mass
ar	aspect ratio
c	specific heat capacity $[J/(kgK)]$
c_{app}	apparent heat capacity $[J/(kgK)]$
c_p	specific heat capacity at constant pressure $[J/(kgK)]$
H	enthalpy $[J]$
k	thermal conductivity $[W/(mK)]$
m	mass $[kg]$
n	number of simulation/measurement values
p	pressure $\left[N/m^2\right]$
Q	heat storage capacity $[J]$

q	heat flux $[W/m^2]$
s	melting front location $[m]$
St_L	Stefan number
T	temperature $[K]$
t	time $[s]$
u	velocity component in x direction $[m/s]$
v	velocity component in y direction $[m/s]$
x, y	space coordinates $[m]$
$X_{aluminium}$	cavity aluminium containment width $[m]$
X_{PCM}	cavity PCM dimension in x direction $[m]$
X_{steel}	steel containment width $[m]$
$Y_{aluminium}$	cavity aluminium fin width $[m]$
Y_{PCM}	cavity PCM dimension in y direction $[m]$
B	derivative of shape functions N
N	shape functions
Subscripts	
0	initial
boundary	value at the boundary
el	number of elements
in	input value
liquid	material parameter of liquid phase
m	melting point
max	maximum value
n	number of nodes
out	output value
ref	reference value
solid	material parameter of solid phase
Superscripts	
*, **	intermediate solutions of the fractional step approach
h	horizontal averaged/differenced quantity
v	vertical averaged/differenced quantity

Abbreviations

CFD	computational fluid dynamics
CFL	Courant-Friedrichs-Lewy (condition)
FDM	finite difference method
FEM	finite element method
FVM	finite volume method
HyStEPs	hybrid storage for efficient processes
PCM	phase change material
RMSE	root mean square error
RSS	Ruths steam storage

Symbols

Δ	finite difference/Laplace operator $\Delta = \nabla^2$
∇	nabla operator: $\nabla = (\partial/\partial x, \partial/\partial y)$

List of Figures

List of Tables

List of code Listings

Chapter 1

Introduction

1.1 Motivation

Climate change and the management of the earth's resources are two major challenges of our time. The progressive decarbonisation process and the increased share of renewable energy sources in the grid has increased the future demand for thermal energy storage in existing industrial plants and processes is required in the future. The efficiency of industrial processes could be increased by balancing steam production and consumption. The Ruths steam accumulator is a well-established thermal storage technology in a variety of industries, including the steel industry.

In order to increase the storage capacity of such steam accumulators, an innovative hybrid storage concept is currently being developed by the HyStEPs project (Hybrid storage for efficient processes), to which this diploma thesis makes a contribution. HyStEPs is a project funded by the Austrian Research Promotion Agency (FFG) with grant number 868842, as part of the Climate and Energy Fund KLI.EN. The project includes the project coordinator Austrian Institute of Technology (AIT) and the partners TU Wien, voestalpine Stahl Donawitz GmbH and EDTMAYER Systemtechnik GmbH.

The basic approach of HyStEPs is to encase Ruths steam storages in innovative latent heat accumulators. Furthermore, a state of charge measurement for the hybrid storage tank should ensure optimal control and operation of both storage types during charging and discharging. This innovative solution is based on an AIT patent application.

1.2 Problem statement

Recently, the applications for latent heat thermal energy storages have increased substantially. The main advantage of these thermal energy storages is the high storage density and the ability to store energy at a nearly constant temperature level, using phase change materials (PCMs). This reduces thermal losses compared to sensible heat storages, since energy can be stored at a lower temperature levels. A large amount of energy is absorbed or released during the phase change of such materials, most commonly between the liquid and solid state. The main disadvantage is the low thermal conductivity of most PCMs, which necessitates various methods to increase the heat transfer between the PCM and the heat transfer medium. One approach is the addition of fins inside the PCM containing cavities built from materials with high thermal conductivity, such as aluminium.

In the HyStEPs project, a number of different design options for encasing cylindric Ruths steam accumulator with a latent heat material have been investigated and are still open for discussion. The most promising of these options, and one that is relatively easy to build, is that of PCM containers with aluminium fins in the plane of the cylinder axis. A schematic illustration of this hybrid storage concept is shown in Figure 1.1. It should be noted that a design optimization of hybrid storage concepts for industrial applications was developed at the same time this thesis was conducted. This optimization was published by fellow HyStEPs project researchers Hofmann et al. [1].

Since heat is transferred in the PCM by both heat conduction and natural convection in the liquid regions, the development of the melting front in the latent heat storage cavity can take very complex forms. This leads to a rather difficult characterisation of the charging and discharging behaviour of the hybrid storage. To determine the position(s) of the melting front(s) in the cavity, a model must be developed which helps to finalize decisions regarding dimensioning and placement of sensors.

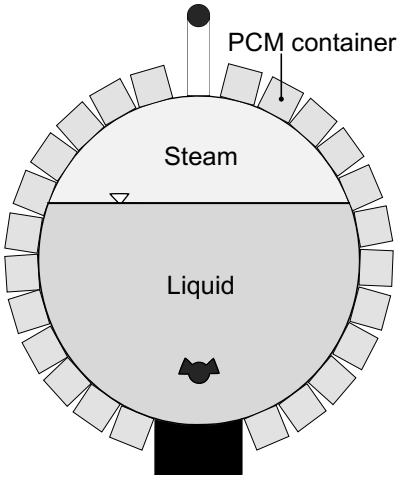


Figure 1.1: Schematic sketch of the hybrid storage design consisting of the Ruths steam storage and the surrounding PCM modules [2]

1.3 Aim of this work

As mentioned above, one of the challenges in the HyStEPs project is the complex charging and discharging behaviour of the PCM material in the cavities mounted to the Ruths steam storage. As contribution to this project, a numerical model is developed in the scope of this diploma thesis to simulate the evolution of melting and solidification during possible operation scenarios.

The purpose of this model is, on the one hand, the prediction of the exact behaviour of the latent heat storage for different operation scenarios and design options. This aids the development of the final HyStEPs storage construction, as well as the choice of PCM material, which should also have optimal characteristics in regards to the operational cycle of the steam storage.

On the other hand, the model should provide a basis for developing a reduced order model, which is part of the operation and control strategy of the HyStEPs project and allows full exploitation of the potential of the latent heat storage. The approach for

this purpose is a model predictive control based on state of charge measurements in the PCM. Thus, optimal sensor placement must be evaluated, which could be made easier by using a reduced model developed on the basis of the fundamental model created in this thesis.

In this work, a preliminary parameter study was conducted to estimate the charging/discharging speed of different PCM cavity designs and to study the effect of natural convection in this specific application.

In comparison to commercially available code, the MATLAB model developed should offer high usability in regards to running and evaluating parameter studies, which can be run parallel on any desired number of computing units. Furthermore, modifications to the existing code should be easy to make, aiding model reduction by using MATLAB directly.

1.4 Methodological approach

The stated problem was analysed with regard to mathematical description, approximation possibilities and numerical modelling options. To this end, a thorough study of relevant academic literature was undertaken. A good review of these topics with regard to phase change materials can be found here [3].

The chosen discretization method of the continuous model is the finite element method, which is well established in applications in structural mechanics, as well as fluid- and thermodynamics. After setting up the fundamental model equations, using temperature as the dependent variable and the apparent heat capacity method to account for the latent heat of phase change, the finite element approximation was derived.

The modelling of the problem stated in Section 1.2 was done in MATLAB [4], which presented a numerical computing environment with very efficient matrix manipulation operations. This is especially suitable for the finite element method.

The model was then validated using an analytical solution for the heat conduction problem and experimental data taken from literature for the problem considering natural convection.

Furthermore, to investigate different aspects of the latent heat storage cavity of the HyStEPs hybrid storage, simulations were carried out in which certain geometry paramaters are varied and their effect on overall behaviour is analysed. Thus, qualitative statements can be made and the here used methodology used can act as a reference for future investigations.

1.5 Structure of this thesis

After this short introduction into the thematics of the problem, this diploma thesis is structured as follows.

In Chapter 2, the underlying theoretical framework for the development of a numerical model describing phase change problems is given. This summary, based on literature review, should provide the reader of this thesis with the most essential knowledge to comprehend the documentation in the subsequent chapters.

The assumptions for the development of the computational model are explained in Chapter 3, where the discretized equations and boundary conditions are also derived. These were later implemented in the form of a MATLAB model, which is documented in Chapter 4. Therein, the basic structure of the code and implementation approaches are explained, and Section 4.3 acts as a guide on how to handle the final version of the code. Detailed instructions on how to specify simulation parameters are available, facilitating use of the code by unfamiliar users.

Chapter 5 deals with the validation of the developed model, which is presented in three steps: The comparison of simulation results considering only heat conduction with the analytical solution of the Stefan problem, the validation of the convection model by experimental data [5], and the cross-validation of the full model with the CFD software ANSYS Fluent. In this chapter of the thesis, very good agreement with

the comparative data was obtained and the developed model was therefore successfully validated. In addition, simulations were carried out for different values of the computational parameters and subsequently analysed. The effect of the chosen mesh size, time step size and mushy region temperature range are discussed, as are possible complications and how to avoid them.

Chapter 6 is a parameter study of the HyStEPs PCM cavity. Within the scope of this thesis, a variation of different cavity dimensions was simulated and their characteristics evaluated. Furthermore, the effect of natural convection was investigated for different orientations with respect to position on the steam storage.

The diploma thesis is concluded in Chapter 7, which includes a short summary of the conducted tasks and the obtained results. The scope and limitations of the developed MATLAB model, and of the assertions gained thereby, are discussed in Section 7.2. Finally, improvements and future research objectives are suggested in Section 7.3.

Chapter 2

Theoretical framework

2.1 Basics of thermodynamics

2.1.1 Principles of thermal energy storage systems

Thermal energy can be stored as a change in the internal energy of a material as either sensible heat, latent heat or thermochemical heat, or a combination of these. Sensible heat storage utilizes the heat capacity and the temperature difference of the material during the process of charging and discharging. The amount of heat stored

$$Q = \int_{T_i}^{T_f} mc_p dT \tag{2.1}$$

depends on the specific heat capacity c_p at constant pressure p of the material, the temperature difference $(T_f - T_i)$ and the mass of the storage material m, under the assumption of constant c_p.

Latent heat storage systems are based on the absorption or release of thermal energy when the storage phase change material (PCM) undergoes a phase change. The heat storage capacity is then given by

$$Q = \int_{T_i}^{T_m} mc_{p,solid}\, dT + ma_m \Delta l_m + \int_{T_m}^{T_f} a_m mc_{p,liquid}\, dT\ , \tag{2.2}$$

with the specific latent heat Δl_m and the liquid fraction of the total mass a_m [6]. The specific heat capacity at constant pressure c_p will simply be denoted as c, or also as c_{solid} for the constant heat capacity of a solid medium and as c_{liquid} for the constant heat capacity of a liquid medium.

Phase transitions occur in different forms:

- **Solid-solid** transitions occur where energy is stored as a material is transformed from one crystalline phase to another. These transitions generally have small

latent heat and small volume changes and offer the advantages of less stringent container requirements and greater design flexibility [7].

- **Solid-gas** transitions, as well as direct **liquid-gas** transitions, have very high latent heat of phase transition, but their large volumetric expansion during these transitions is associated with containment problems and often rule out their potential utility in thermal energy storage systems [6].
- **Solid-liquid** transformations have comparatively smaller latent heat than liquid-gas transformations. However, these transformations involve only a small change (on the order of 10% or less) in volume and have therefore proven to be economically attractive for use in thermal energy storage systems [6].

The phase change from solid to liquid (liquefaction) or liquid to solid (solidification) is preferred for the use case in this thesis because the operating pressure is lower than that of liquid to gas or solid to gas phase changes. From here on, this thesis discusses exclusively the solid to liquid or liquid to solid type of phase change.

Liquefaction and solidification can either occur at a constant melting temperature (isothermal phase change) or in a temperature range around the melting temperature T_m, which is often called a mushy phase change. Numerous materials, especially pure crystalline substances and eutectics show a sharply defined value for the melting temperature T_m [8], but many PCM materials show a significant melting range. This aids some numerical models in avoiding discontinuities, which can cause problems, as we will see later in this thesis. The change of enthalpy during the phase change for these two types is illustrated in Figure 2.1.

2.1.2 Heat transfer mechanics

In a solid medium, heat is transferred via heat conduction by microscopically colliding particles, including molecules, atoms and electrons, which shift their internal energy from one particle to another. The rate of the energy transfer can be shown as a

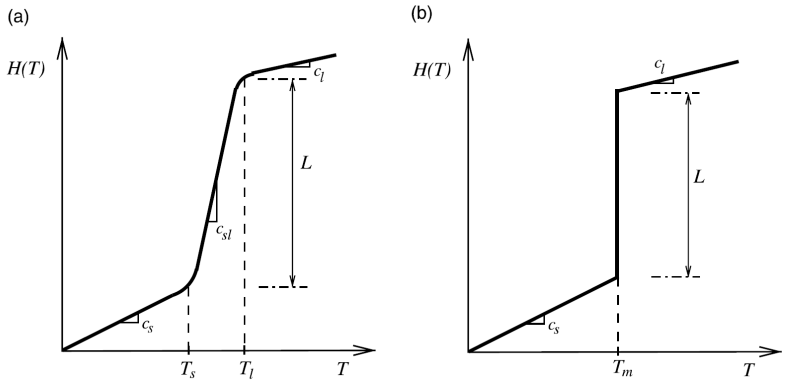


Figure 2.1: Enthalpy H versus temperature T with (a) mushy phase change and (b) isothermal phase change [9]

Note. L denotes the specific latent heat.

function of the temperature difference (temperature gradient) in the medium and its conductive properties. This basic law, also known as Fourier's law, is expressed as

$$\mathbf{q} = -k\nabla T \ , \tag{2.3}$$

where $\vec{q}$ is the thermal flux and k is the thermal conductivity of the medium [10]. In a liquid medium heat is transferred both by conduction and convection. Convection is the transfer of thermal energy via the movement of fluid, and is usually driven by pressure differences in the liquid. In the absence of outer forces, so-called natural convection currents occur, due to temperature-triggered density variations. This leads to a thermal flux

$$\mathbf{q} = \rho c T \mathbf{u} \ , \tag{2.4}$$

where ρ is the density of the liquid and $\mathbf{u}$ is the velocity of the liquid.

The energy equation for transient heat transfer by both conduction and convection [11] is

$$\frac{\partial(\rho c T)}{\partial t} = \nabla(k\nabla T) - \nabla(\rho c T \cdot \mathbf{u}) \ . \tag{2.5}$$

According to the literature, this equation is also called the convection-diffusion equation or advection-diffusion equation, depending on the context [12]. Aside from the conservation of energy (2.5), conservation of mass and momentum must be fulfilled.

$$\frac{\partial(\rho)}{\partial t} = \nabla(\rho \mathbf{u}) \tag{2.6}$$

For an incompressible fluid with $\frac{\partial(\rho)}{\partial t} = 0$, this continuity equation takes the form

$$\nabla(\rho \mathbf{u}) = 0 \ . \tag{2.7}$$

The momentum balance, also known as the Navier-Stokes-equation, takes the following form for an incompressible Newtonian fluid with constant material properties $\mu = const., \rho = \rho_0 = const.$:

$$\rho_0 \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) - \nabla \left(\mu \nabla \mathbf{u} \right) + \nabla p = \mathbf{f} \ [13] \ , \tag{2.8}$$

including the pressure p, the dynamic viscosity μ of the liquid and the external force density $\mathbf{f}$, which usually only consists of the density and the gravitational acceleration $\mathbf{g}$.

2.1.3 Boundary conditions

To solve a system of differential equations, boundary conditions must be established, as must an initial condition for the solution. The main types of boundary conditions used in thermodynamics for the temperature $T(\Omega, t)$ at a boundary $\delta\Omega$ of the domain Ω are listed below [14].

- **Dirichlet** boundary conditions prescribe a temperature at the boundary:

$$T(\delta\Omega, t) = T_{boundary}(t) \ . \tag{2.9}$$

- **Neumann** boundary conditions prescribe a heat flux $\mathbf{q}_{boundary}(t)$ at the boundary. By using Fourier's law (2.3) these can be written as

$$-k \frac{\partial T}{\partial \mathbf{n}} \ |_{\delta\Omega,t} = \mathbf{q}_{boundary}(t) \ , \tag{2.10}$$

where $\mathbf{n}$ is the outward unit normal to the boundary surface [9]. For perfectly insulated boundaries (adiabatic wall) this means

$$-k\frac{\partial T}{\partial \mathbf{n}} \mid_{\delta\Omega,t} = 0 \ . \tag{2.11}$$

- **Robin** boundary conditions prescribe a heat exchange with the surrounding boundary at the ambient temperature $T_{boundary}(t)$ with a specified thermal conductivity $\alpha_{boundary}$:

$$-k\nabla T \mid_{\delta\Omega,t} = \alpha_{boundary} \cdot (T \mid_{\delta\Omega,t} - T_{boundary}(t)) \ . \tag{2.12}$$

  This type of boundary condition is often also called Fourier condition or convective boundary condition [9].

It should be noted that there are more types of boundary conditions which are not relevant for this thesis and will therefore not be discussed. One example would be radiation boundary conditions when considering heat radiation effects or periodic boundary conditions, which are useful for symmetric problems. Combinations of the boundary condition types mentioned are sometimes used.

The most common type of boundary condition for the velocity $\mathbf{u}(\Omega, t)$ is the **no-slip** condition

$$\mathbf{u} \mid_{\delta\Omega,t} = 0 \ , \tag{2.13}$$

which essentially prescribes that the fluid at the boundary has zero velocity relative to the boundary, which implies zero velocity in case of a resting boundary. However, for low pressure or high velocity, as well as for low viscosity fluids, a full-slip or partial-slip condition could prove more accurate [13].

2.1.4 The Stefan problem

The analytical solution of the system of differential equations given by (2.5), (2.7) and (2.8) in combination with certain initial and boundary conditions can only be found for a small class of problems. In general, this boundary value problem must be solved numerically, as discussed in Section 2.2.

The simplest form of a mathematical model describing phase transitions is the classic, one-dimensional Stefan problem. This problem describes the liquefaction of a solid medium, initially at the solidification temperature $T(x, t = 0) = T_m$ with the boundary condition $T(x = 0, t) = T_0 > T_m$. To acquire the analytical solution to the classic Stefan problem, meaning the temperature distribution in the liquid phase and the location of the melting front s(t), the heat equation without convection must be solved. A standard form of the problem can be declared as follows [15]:

The liquid region	$0 \leq x < s(t)$
$\frac{\partial(\rho c T(x,t))}{\partial t} = \nabla(k \nabla T(x,t))$	heat equation, $0 < x < s(t)$, $t > 0$
$T(x = 0, t) = T_0 > T_m$	Boundary condition, $t > 0$
$T(x, t = 0) = T_m$	Initial condition
The free boundary	$x = s(t)$
$\Delta l_m \rho \frac{\partial s}{\partial t} = -k \frac{\partial T}{\partial x}$	Stefan condition
$s(t = 0)$	Initial position of melting front
$T(s(t), t) = 0$	Dirichlet condition at the melting front
The solid region	$s(t) < x < \infty$
$T(x, t)$	t, $x \geq s(t)$

A similarity solution [15] of this problem can be found below, dependent on the Stefan number $St_L = \frac{c(T_0 - T_m)}{\Delta l_m}$ and the thermal diffusivity $\alpha_L = \frac{k}{c\rho}$

$$T(x,t) = T_0 - \frac{T_0}{erf(\lambda_{St})} \cdot erf(\frac{x}{2\sqrt{\alpha_L t}}) \tag{2.14}$$

$$s(t) = 2\lambda_{St}\sqrt{\alpha_L t} \tag{2.15}$$

$$\lambda_{St} e^{\lambda_{St}^2} erf(\lambda_{St}) = \frac{St_L}{\sqrt{\pi}} \ . \tag{2.16}$$

Here, λ_{St} is a parameter defined by the transcendental equation (2.16), dependent on the Stefan number St_L. This solution for the location of the melting front (2.15) will later be used to validate the developed numerical model of the one-dimensional heat equation.

2.2 Numerical methods

To solve the system of differential equations given by (2.5), (2.7) and (2.8), numerical methods to accurately describe the phase change must be implemented. Furthermore, the Stefan problem is complicated by the fact that, in general, more than one moving phase boundary can occur in PCM ([16]).

Different approaches to this problem have been developed, the most common of which are reviewed in the article [17] on which the following section is based.

2.2.1 Enthalpy method

One of the most popular approaches is the enthalpy method which can be illustrated by considering a one-dimensional isothermal phase change problem controlled only by heat conduction. The enthalpy function H is defined as the integral of the volumetric heat capacity with respect to temperature:

$$H(T) = \Delta H(T) + \rho f(T) a = \int_{T_{ref}}^{T} \rho c(T) dT + \rho f(T) a \tag{2.17}$$

The first term of the enthalpy function (2.17) accounts for the sensible heat of the material, while the second term accounts for the latent heat of phase change. To obtain the total enthalpy for a temperature T, the liquid fraction

$$a = \begin{cases} 0 & \text{, solid: } T < T_m \\ ]\,0,1\,[& \text{, mushy: } T = T_m \\ 1 & \text{, liquid: } T > T_m \end{cases} \tag{2.18}$$

and the function $f(T)$, describing the amount of latent heat at the temperature T, must be given. This leads to an alternate form of the energy equation (2.5)

$$\frac{\partial \Delta H}{\partial t} = \frac{\partial}{\partial x}\left(\alpha_L \frac{\partial \Delta H}{\partial x}\right) - \rho a \frac{\partial f}{\partial t}\ , \tag{2.19}$$

with the thermal diffusivity $\alpha_L = \frac{k}{c\rho}$.

The enthalpy method leads to a simple phase change problem, since interface conditions must not be taken into account [17].

2.2.2 Apparent heat capacity method

In some applications, it is more suitable to use temperature rather than enthalpy as the dependent variable. In these cases, the apparent heat capacity method is often chosen to model the phase change problem. Here, the governing equation is the energy equation (2.5), but the apparent heat capacity of the material c_{app} is modelled to account for the latent heat during the phase change:

$$c_{app} = \begin{cases} c_{solid} & \text{, solid: } T_m - \varepsilon > T \\ \frac{\Delta l_m + c_{solid} \cdot (T_m + \varepsilon - T) + c_{liquid} \cdot (-T_m + \varepsilon + T)}{2\varepsilon} & \text{, mushy: } T_m - \varepsilon \leq T \leq T_m + \varepsilon \\ c_{liquid} & \text{, liquid: } T > T_m + \varepsilon \ . \end{cases} \tag{2.20}$$

This method obviously requires the existence of a phase change temperature range, also called melting range, of the width 2ϵ around the melting temperature T_m. In modelling phase change problems with small values of ϵ, this can lead to problems [11]. In addition, the method is generally not applicable in cases with isothermal phase change. However, it is possible to approximate isothermal phase change problems by introducing an appropriately small artificial mushy region of the width 2ϵ.

2.3 Discretization methods

The basic premise of a discretization method is to replace the original continuous problem by a sequence of finite-dimensional problems [18]. In the past, several methods of discretization have been developed to numerically solve problems of partial differential equations, the most common on which will be explained here. It should be noted that there is a distinction between fixed grid and adaptive grid methods, both of which have advantages for specific problems. In this thesis, only fixed grid methods are used and therefore explained, however, most of the following principles can be employed for adaptive grid methods as well.

In the next sections the main discretization methods with respect to space will be described, although most problems are also continuous in time and therefore also

need to be discretized appropriately. Time discretization methods will be outlined in Section 2.3.4.

2.3.1 Finite element method

The first step in any finite element (FEM) simulation is to define a grid of the continuous computational domain. A grid consists of a finite number of non-overlapping elements that cover the whole domain [19]. The most common geometries of elements are triangular elements and quadratic elements, the latter of which are used in the model developed in this work. The connection points between the elements, typically located on the corners of the element, are called nodes. Of course, it is also possible to define elements with more nodes, which enhances the accuracy of the approximation, but requires more computational effort.

Based on the defined grid, or mesh, continuous variables, such as the temperature T, can be decomposed by the shape functions $[N] = [N1, N2, N3...]$ and temperatures at the element nodes $\{T\} = \{T1, T2, T3...\}$:

$$T = [N] \cdot \{T\} \ . \tag{2.21}$$

The general approach of the finite element method is to multiply the partial differential equation (PDE) with given boundary conditions (the strong form) by a test function, or weighting function, and integrate over the whole simulation domain. Applying Green's first integration theorem (partial integration) leads to a variational formulation, also called the weak form [20]. To discretize this still infinite dimensional equation, the approximation of continuous variables, as in (2.21), is applied to the weak form. In the Galerkin procedure, which will also be used in this thesis, the weighting functions are equal to the shape functions, which leads to a system of equations with the same number of equations as unknowns, that is the number of nodes [19]. In the case of a stationary problem, this is a system of algebraic equations, otherwise this becomes a problem of ordinary differential equations.

Although finite difference (FDM) and finite volume schemes (FVM) are well established in computational fluid dynamics (CFD) applications, including the well-known

software package ANSYS Fluent, Nedjar [9] states that especially finite element methods (FEM) are able to handle complex coupled thermomechanical problems with various and complex boundary conditions.

2.3.2 Finite difference method

The finite difference method (FDM) can also use the same grid as in 2.3.1. The basic principle of FDM is that the derivatives in the partial differential equation are approximated by linear combinations of function values at the grid points x_i. For first order derivatives, the difference quotient of a variable $u(x)$ is by definition [21]

$$\frac{\partial u}{\partial x}(x) = \lim_{\Delta x \to 0} \frac{u(x+\Delta x) - u(x)}{\Delta x} = \lim_{\Delta x \to 0} \frac{u(x) - u(x-\Delta x)}{\Delta x} = \lim_{\Delta x \to 0} \frac{u(x+\Delta x) - u(x-\Delta x)}{2\Delta x} \tag{2.22}$$

for an infinitesimal difference Δx between the grid points. This leads to three types of commonly used approximations:

$$\begin{aligned} \left[\frac{\partial u}{\partial x}\right]_i &\approx \frac{u_{i+1} - u_i}{\Delta x} && \text{forward difference} \\ \left[\frac{\partial u}{\partial x}\right]_i &\approx \frac{u_i - u_{i-1}}{\Delta x} && \text{backward difference} \\ \left[\frac{\partial u}{\partial x}\right]_i &\approx \frac{u_{i+1} - u_{i-1}}{2\Delta x} && \text{central difference .} \end{aligned} \tag{2.23}$$

The derivatives, which typically arise in two-dimensional convection-diffusion problems for the variable $u(x, y)$ and the corresponding grid point indices i, j are calculated for the central difference method, as can be seen below:

$$\left[\frac{\partial u}{\partial x}\right]_{i,j} \approx \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} \tag{2.24}$$

$$\left[\frac{\partial u}{\partial y}\right]_{i,j} \approx \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y} \tag{2.25}$$

$$\left[\frac{\partial^2 u}{\partial x^2}\right]_{i,j} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{4\Delta x^2} \tag{2.26}$$

$$\left[\frac{\partial^2 u}{\partial x \partial y}\right]_{i,j} \approx \frac{u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}}{4\Delta x \Delta y} \; . \tag{2.27}$$

The derivative of a nonlinear term $u(x,y)\cdot v(x,y)$ can be calculated via the product rule thus

$$\left[\frac{\partial uv}{\partial y}\right]_{i,j} = u_{i,j}\left[\frac{\partial v}{\partial y}\right]_{i,j} + v_{i,j}\left[\frac{\partial u}{\partial y}\right]_{i,j} \approx \frac{u_{i,j}}{2\Delta y}\left(v_{i,j+1} - v_{i,j-1}\right) + \frac{v_{i,j}}{2\Delta x}\left(u_{i+1,j} - u_{i-1,j}\right) \ . \quad (2.28)$$

2.3.3 Finite volume method

Although not implemented in the context of this work, it is worth mentioning another important discretization method, the finite volume method (FVM). As in the finite difference and finite element methods, the first step is the discretization of the computational domain, which in this case means splitting it into non-overlapping elements, or finite volumes. The partial differential equations are transformed into algebraic equations by integrating them over each discrete element, in other words, by taking the integral form of conservation equations and splitting the integrals into small intervals. The finite volume method is strictly conservative, because, since two neighbouring cells share a common interface, the total flux entering a volume is identical to that leaving the adjacent volume [22]. This property makes the FVM the preferred method in computational fluid dynamics problems (CFD) [23].

2.3.4 Time discretization

One of the most used methods for time discretization in numerical problems is the so called θ-method, in which the time derivative of a variable $T(t)$ is replaced by a simple differential quotient

$$\frac{\partial T}{\partial t} = \frac{T^{n+1} - T^n}{\Delta t} \ . \quad (2.29)$$

Here, $T^n = T(x,t_n)$ is the variable's value at $t = t_n$ and Δt is the time increment according to $t_{n+1} = t_n + \Delta t$. The variable $T(x,t_n)$ is usually assumed to be known

and used as an initial condition to advance the solution to the next time level. The solution T can then be written as

$$T = \theta T^{n+1} + (1-\theta)T^n \ , \ t_n \leq t \leq t_{n+1} \ . \tag{2.30}$$

The relaxation parameter $\theta \in [0,1]$ is most commonly one of the following:

$$\begin{array}{ll} \theta = 1 & \text{implicit backwards Euler scheme} \\ \theta = \frac{1}{2} & \text{second-order, centered implicit method (Crank-Nicolson-Galerkin)} \quad [19] \\ \theta = 0 & \text{explicit forward Euler scheme} \end{array} \tag{2.31}$$

It is known that for $\theta < \frac{1}{2}$, only conditional stability is attained [24]. In the model developed in this thesis, the implicit backwards Euler scheme was chosen.

In Chapter 3.2 the semi-discrete Galerkin-formulation

$$[C]\left\{\dot{T}\right\} + [K]\{T\} = [R] \tag{3.23}$$

will be developed (discrete in space, continuous in time). In this equation, the square brackets denote a matrix and the braces denote a vector of node point values.

If we consider this system of ordinary differential equations and apply the backwards Euler scheme, we evaluate the equation at the time $t = t_{n+1}$:

$$[C]\left\{\dot{T}\right\}_{n+1} + [K]\{T\}_{n+1} = [R] \ . \tag{2.32}$$

The time derivative of the temperature vector $\{T\}$ is approximated through the backwards differential quotient

$$\left\{\frac{\partial T}{\partial t}\right\}_{n+1} = \left\{\dot{T}\right\}_{n+1} = \frac{\{T\}_{n+1} - \{T\}_n}{\Delta t} \ , \tag{2.33}$$

which gives us the value

$$\{T\}_{n+1} = \{T\}_n + \Delta t \left\{\dot{T}\right\}_{n+1} \ . \tag{2.34}$$

Combining equation (2.32) with (2.34) leads, with the expression

$$[M] = [C] + \Delta t \cdot [K] \ , \tag{2.35}$$

to

$$[M]\left\{\dot{T}\right\}_{n+1} = ([C] + \Delta t \cdot [K])\left\{\dot{T}\right\}_{n+1} = [R] - [K]\{T\}_n \ . \tag{2.36}$$

We can therefore derive the value for $\left\{\dot{T}\right\}_{n+1}$ as

$$\left\{\dot{T}\right\}_{n+1} = [M]^{-1} \cdot ([R] - [K]\{T\}_n) \ , \tag{2.37}$$

if the matrix $[M]$ (equation (2.35)) is invertible and nonsingular [20]. The solution for $\{T\}_{n+1}$ can thus be found through equation (2.34) to be

$$\begin{aligned} \{T\}_{n+1} = & \{T\}_n + \Delta t \left\{\dot{T}\right\}_{n+1} \\ & \{T\}_n + \Delta t\,[M]^{-1} \cdot ([R] - [K]\{T\}_n) = \\ & \{T\}_n + \Delta t\,([C] + \Delta t \cdot [K])^{-1} \cdot ([R] - [K]\{T\}_n) \ . \end{aligned} \tag{2.38}$$

Chapter 3

Numerical modelling

In this chapter, the underlying assumptions for developing a computational model of the PCM storage are explained, and the discretized equations and boundary conditions are derived.

Although a number of different design options are currently being discussed for the PCM storage containers attached to the cylindric Ruth steam storage, one of the most promising and easiest to build calls for PCM containers with aluminium fins in the plane of the cylinder axis. For general fin arrangements, a three-dimensional cavity simulation might be required. However, the geometry of the PCM cavities regarded in this work, in which the depth of the enclosure parallel to the cylinder axis is assumed large enough for wall boundary layer effects to be negligible, suggests symmetry reduction of the simulation problem to single aluminium fin segments and a two-dimensional approach. In the hybrid storage concept in [25] this geometry relates to an aluminium fin structure orientated in the radial-axial plane of an RSS.

This sort of design is modelled in this thesis, although the developed model can be easily modified to suit similar problems. Unfortunately, it is not possible to simulate a design with fins oriented in a plane perpendicular to the cylinder axis. Heat conduction and convection effects would make this a three-dimensional problem and, as such, this would require further extensions to a three-dimensional finite element model.

3.1 Model equations and assumptions

3.1.1 Geometry

Although the fins are not exactly parallel, the diameter of the cylindric Ruths steam storage is assumed much larger than the thickness of the attached PCM modules. Therefore, it is reasonable to see the PCM container as rectangular. Also, it is possible that the containers will later be constructed in a rectangular design, due to their simplified construction. The underlying equations of the problem will therefore be developed for a rectangular mesh of finite elements. PCM material, aluminium fins, as well as containment or other materials can later be defined separately for each element.

3.1.2 Material properties

Currently, several different PCM materials are being discussed for inclusion in the application described in this work. A comparison of different PCM materials can be found in [26]. Thus far, the most promising material considering the container geometry and operating temperatures of this hybrid storage system is a eutectic mixture of potassium nitrate and sodium nitrate (KNO_3-$NaNO_3$). The properties of this mixture, as well as those of a typical containment steel, carbon steel 1.0425, taken from reference [27], are compared in Table 3.1. In this reference, it is noted that despite the theoretical melting temperature of the mixture at 222 °C, the onset of melting of the technical grade material used was measured at 219.15 °C. For the purpose of basic evaluations in this thesis, an approximation of $T_m = 220\,°C$ is used and is therefore given as such in Table 3.1.

There is no exact number to declare the melting temperature range of (KNO_3-$NaNO_3$), but there are results of different measurements found in literature. In reference [28], the reported result for the onset of melting was $220.6 \pm 0.3\,°C$ and for the onset of crystallization $223 \pm 0.34\,°C$, which would result in a melting range of

around 2.5 °C. It was also noted that the results are in very good agreement with those found in other literature and that neither subcooling nor thermochemical instability have been observed [28].

Table 3.1: Thermophysical material properties

Note. (s) denotes the solid phase, (l) denotes the liquid phase.

Material	ρ / $kg\,m^{-3}$	c / $J(kg\,K)^{-1}$	k / $W(m\,K)^{-1}$	T_m / $^{\circ}C$	Δl_m / $kJ\,kg^{-1}$	β / K^{-1}	μ / $N\,s\,m^{-2}$
KNO_3-	2050(s)	1350(s)	0.457(s)	220	108	$3.5 \cdot 10^{-4}$	$5.8 \cdot 10^{-3}$
$NaNO_3$	1959(l)	1492(l)	0.435(l)				
Steel	7800	540	51	-	-	-	-
Aluminium	2700	910	237	-	-	-	-
Air	1	1000	0.024	-	-	-	-

For most simulations aiming to assess the behaviour of the potential PCM container, a mushy region of 2 °C was introduced, otherwise this will be declared. The effect of the mushy region with respect to possible numerical difficulties will be discussed later in this thesis.

Also given in Table 3.1 are the well known properties of aluminium and air [29], which are used for certain simulations.

Regarding the heat transfer from the RSS to the PCM containers, the choice of inner heat transfer coefficient is not straightforward, since it is highly dependent on the inner thermodynamic state in the RSS and therefore on the operating scenario. If the RSS is charged, condensing steam dominates the heat transfer to the outer wall in the vapour phase, while in the liquid phase, the heat transfer is dominated by liquid water. During discharging, heat transfer through boiling water occurs at the outer wall in the liquid phase, and heat transfer through dry steam occurs in the vapour phase [30]. The respective heat transfer coefficients are listed in Table 3.2.

In the parameter studies following in Chapter 6, heat transfer coefficients were assumed that are within the range given here. However, the impact of different heat transfer coefficients on the hybrid storage behaviour was not within the scope of this thesis. A further investigation of this research topic was pursued in [31].

Table 3.2: Heat transfer coefficients α occurring between the inside of the RSS and its outer wall [32]

Operating mode	Phase	Dominating heat transfer effect	$\frac{\alpha}{W\,m^{-2}\,K^{-1}}$
charging	vapour	condensing steam	5000
charging	liquid	liquid water	700
discharging	vapour	dry steam	10
discharging	liquid	evaporating liquid water	1000

3.1.3 Governing equations

Based on the theoretical framework explained in Section 2.1.2, the governing equation for the conduction-convection-model can again be stated as follows:

$$\text{energy equation:} \quad \frac{\partial(\rho c T)}{\partial t} = \nabla(k \nabla T) - \nabla(\rho c T \cdot \mathbf{u}) \tag{2.5}$$

$$\text{continuity equation:} \quad \nabla(\rho \mathbf{u}) = 0 \tag{2.7}$$

$$\text{Navier-Stokes equation:} \quad \rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) - \nabla \left(\mu \nabla \mathbf{u} \right) + \nabla p = \mathbf{f} \tag{2.8}$$

By assuming a constant density ρ over the whole operating temperature range and neglecting the time and space dependency of the heat capactiy c, which is often found in literature for PCM studies using the apparent heat capacity method [11], the energy equation can be written as

$$\rho c \frac{\partial(T)}{\partial t} = \nabla(-\mathbf{q}) - \rho c \nabla(T \cdot \mathbf{u}) \ , \tag{3.1}$$

with the flux $\mathbf{q} = -k\nabla T$. Assuming constant density also means freedom of divergence of the velocity field, thus equation (2.7) is reduced to

$$\nabla \mathbf{u} = 0 \ . \tag{3.2}$$

To account for phase change in the material, the apparent heat capacity method is used, as introduced in Section 2.2.2:

$$c_{app} = \begin{cases} c_{solid} & \text{, solid: } T_m - \varepsilon > T \\ \frac{\Delta l_m + c_{solid} \cdot (T_m + \varepsilon - T) + c_{liquid} \cdot (-T_m + \varepsilon + T)}{2\varepsilon} & \text{, mushy: } T_m - \varepsilon \leq T \leq T_m + \varepsilon \\ c_{liquid} & \text{, liquid: } T > T_m + \varepsilon \end{cases} \tag{2.20}$$

For better readability, the apparent heat capacity c_{app} is henceforth denoted only as c.

The Navier-Stokes equation (2.8) cited in the previous chapter is already in a particular form, that of an incompressible Newtonian fluid with constant material properties ρ, $\mu = const$. A further approximation is to assume constant pressure and only account for density differences in terms of the force density $\mathbf{f}$. This is known as the Boussinesq approximation, where the buoyancy force

$$\mathbf{f} = -\rho \mathbf{g} = -\rho_0 \left(\beta \left(T - T_{ref}\right)\right) \mathbf{g} \tag{3.3}$$

can be calculated by a linear thermal expansion coefficient β, the reference density ρ_0 and a reference temperature T_{ref} [33]. Thus, the Navier-Stokes equation to be solved is

$$\rho_0 \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) - \mu \nabla \left(\nabla \cdot \mathbf{u}\right) + \nabla p = \mathbf{f} \ . \tag{3.4}$$

3.1.4 Boundary conditions

To successfully solve the system of partial differential equations, certain initial and boundary conditions must be given. An initial condition for the temperature in the simulation domain must be specified, and the values can be fixed for each individual node, if desired. Concerning the domain boundary conditions, the following types have been implemented in the model:

- Robin boundary conditions (see (2.12)) have been applied to the east and west domain boundaries:

$$|\mathbf{q}(x = 0, y)| = |\mathbf{q}_{in}(y)| = \alpha_{in} \cdot (T(x) - T_{in}) \text{ and} \tag{3.5}$$

$$|\mathbf{q}(x = L_x, y)| = |\mathbf{q}_{out}(y)| = \alpha_{out} \cdot (T(x) - T_{out}) \ . \tag{3.6}$$

- Adiabatic Neumann boundary conditions have been applied to the south and north domain boundaries:

$$\mathbf{q}(x, y = 0) = 0 \text{ and} \tag{3.7}$$

$$\mathbf{q}(x, y = L_y) = 0 \ . \tag{3.8}$$

The selectable thermal conductivities α_{in} and α_{out} and ambient temperatures T_{in} and T_{out} can also be chosen to be dependent of time and x or y respectively. For the sake of better readability, this is not explicitly depicted here. The values of L_x and L_y denote the length of the domain in the corresponding dimension.

Regarding the velocity field $\mathbf{u} = [u, v]^T$, no slip boundary conditions are set for the domain boundaries

$$\mathbf{u} = [u, v]^T = 0 \ , \text{ for } x = L_x, y = L_y, x = 0, y = 0 \ , \tag{3.9}$$

and the velocity is also set to zero if the PCM is solid, as well as for every non-melting material

$$\mathbf{u} = [u, v]^T = 0 \ , \text{ for } T < T_m - \epsilon \ . \tag{3.10}$$

3.2 Discretization

In the following section, the energy equation (3.1) is discretized using the standard Galerkin finite element method, as introduced in Section 2.3.1. The time discretization implemented later is chosen as the implicit backwards Euler scheme, as explained in section 2.3.4.

The Navier-Stokes equation (3.4) is treated separately via a finite difference discretization, which will be explained in Section 3.3.

3.2.1 Preliminary remarks

The two-dimensional domain ($x \in [0, L_x]$, $y \in [0, L_y]$) is split into an equidistant mesh of $n_{nel} = n_x \cdot n_y$ finite elements. In this thesis, four-noded bilinear rectangular elements are used, which leads to the total sum of $n_{non} = (n_x + 1) \cdot (n_y + 1)$. The dependent variables, meaning the temperature T and the velocity $\mathbf{u} = [u, v]^T$, can therefore be approximated by their nodal values as

$$T = [N] \cdot \{T\} \ , \tag{3.11}$$

$$u = [N] \cdot \{u\} \text{ and} \tag{3.12}$$

$$v = [N] \cdot \{v\} \ , \tag{3.13}$$

where $[N] = [N_1, N_2, N_3, ..., N_{non}]$ is the vector of shape functions and the braces denote a vector of node point values. In the notation used here, general matrices are represented by square brackets.

The flux can be discretized as

$$\mathbf{q} = -k \cdot [B] \cdot \{T\} \tag{3.14}$$

when using the gradient of the shape function matrix

$$[B] = \begin{bmatrix} \dfrac{\partial N1}{\partial x} & \dfrac{\partial N2}{\partial x} & \dfrac{\partial N3}{\partial x} & \cdots \\ \dfrac{\partial N1}{\partial y} & \dfrac{\partial N2}{\partial y} & \dfrac{\partial N3}{\partial y} & \cdots \end{bmatrix} . \tag{3.15}$$

To illustrate the approximation via the four-noded bilinear elements, we consider a finite element in the natural coordinate system ξ, η with the length $dx = 2$ and height $dy = 2$ (see Figure 3.1). A function $\Phi(\xi, \eta)$ can then be approximated in terms of the nodal values and shape functions as

$$\Phi(\xi, \eta) = N_1\Phi_1 + N_2\Phi_2 + N_3\Phi_3 + N_4\Phi_4 \ , \tag{3.16}$$

with the shape functions

$$\begin{aligned} N_1 &= \frac{1}{4}(1-\xi)(1-\eta) \ , \\ N_2 &= \frac{1}{4}(1+\xi)(1-\eta) \ , \\ N_3 &= \frac{1}{4}(1+\xi)(1+\eta) \ , \\ N_4 &= \frac{1}{4}(1-\xi)(1+\eta) \ [19] \ . \end{aligned} \tag{3.17}$$

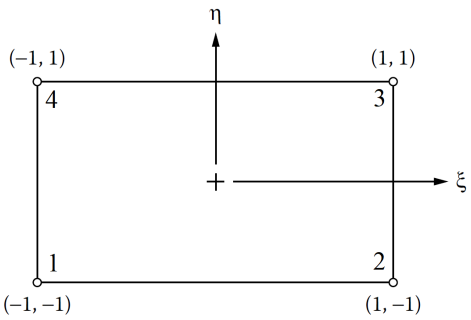


Figure 3.1: Four-noded bilinear element in natural coordinate system ξ, η [19] with node coordinates in brackets

3.2.2 Galerkin's method

Using Galerkin's method on the energy equation (3.1) leads to

$$\int_V [N]^T \rho c \frac{\partial T}{\partial t} \, dV + \int_V [N]^T \nabla(\mathbf{q}) \, dV + \int_V [N]^T \rho c \nabla(T\mathbf{u}) \, dV = 0 \ . \tag{3.18}$$

When applying the divergence theorem $\int_V \nabla F \, dV = \int_S F \cdot \mathbf{n} \, dS$ [34] on the third integral we obtain the following relation:

$$\begin{aligned} \int_V [N]^T \rho c \nabla(T \cdot u) \, dV + \int_V \nabla([N]^T) \rho c T \mathbf{u} \, dV = \int_V \rho c \nabla([N]^T \, T\mathbf{u}) \, dV = \\ \int_S \rho c ([N]^T \, T\mathbf{u})^T \cdot \mathbf{n} \, dS \ . \end{aligned} \tag{3.19}$$

Because of boundary condition (3.9), the surface integral over the domain boundary is equal to zero and the third integral of equation (3.18) can be written as:

$$\int_V [N]^T \, \rho c \nabla(T\mathbf{u}) \, dV = \int_V \nabla([N]^T) \rho c T \mathbf{u} \, dV \ . \tag{3.20}$$

Similarly, inserting the flux $\mathbf{q} = -k\nabla T$ into the second integral and applying the divergence theorem leads to

$$\int_V [N]^T \, \nabla(\mathbf{q}) \, dV = \int_V \nabla([N]^T) k \nabla T \, dV + \int_S [N]^T \, \mathbf{q} \cdot \mathbf{n} \, dS \ , \tag{3.21}$$

which is written as

$$\begin{aligned} \int_V [B]^T \, k \, [B] \, \{T\} \ dV + \int_{S1} [N]^T \, \alpha_{out} ([N] \, \{T\} - T_{out}) \, dS + \\ \int_{S3} [N]^T \, \alpha_{in} (T_{in} - [N] \, \{T\}) \, dS + \int_{S2} 0 \, dS + \int_{S4} 0 \, dS \end{aligned} \tag{3.22}$$

when applying the boundary conditions for the boundary S, where the index $S1$ denotes the east boundary, $S3$ the west boundary and indices $S2$, $S4$ the south and north boundaries, where zero flux is defined.

Finally, inserting the finite element approximations into equation (3.18) leads to the system of ordinary differential equations

$$[C] \left\{ \dot{T} \right\} + [K] \, \{T\} = [R] \ , \tag{3.23}$$

with the following system matrices:

$$[C] = \int_V [N]^T \, \rho c \, [N] \ dV \ , \tag{3.24}$$

$$\begin{aligned} [K] = \int_V [B]^T \, \mathbf{k} \, [B] \ dV \ - \int_V ([B]^T) \rho c \mathbf{u} \, [N] \ dV \\ + \int_{S1} \alpha_{out} ([N]^T \, [N] \ dS - \int_{S3} \alpha_{in} ([N]^T \, [N] \ dS \ , \end{aligned} \tag{3.25}$$

$$[R] = \int_{S1} \alpha_{out} T_{out} \, [N]^T \ dS - \int_{S3} \alpha_{in} T_{in} \, [N]^T \ dS \ . \tag{3.26}$$

The system matrix $[R]$ is of the dimension $non \times 1$, while the matrices $[C]$ and $[K]$ are of the dimension $non \times non$, when the velocity $\mathbf{u} = [u, v]^T$ and the thermal conductivity $\mathbf{k} = [k_x, k_y]^T = k \cdot [1, 1]^T$ are evaluated. The time discretization of equation (3.23)

is done by application of the implicit backwards Euler method as derived in Section 2.3.4 of this thesis:

$$\begin{aligned} \{T\}_{n+1} &= \{T\}_n + \Delta t \left\{\dot{T}\right\}_{n+1} \\ &\{T\}_n + \Delta t \left[M\right]^{-1} \cdot \left(\left[R\right] - \left[K\right]\{T\}_n\right) = \\ &\{T\}_n + \Delta t \left(\left[C\right] + \Delta t \cdot \left[K\right]\right)^{-1} \cdot \left(\left[R\right] - \left[K\right]\{T\}_n\right) \ . \end{aligned} \tag{2.38}$$

3.2.3 Integration scheme

To numerically compute the system matrices derived in the previous section, the integration is split into the finite domains of the defined elements. The integrals are first transformed into the natural coordinate system $1 \leq \xi \leq \eta \leq 1$, which leads to simple integration limits:

$$\int_V f(x,y)\, dV = \int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta)\, |\mathbf{J}|\, d\xi d\eta \tag{3.27}$$

The factor $|\mathbf{J}|$ is the determinant of the Jacobian matrix of the transformation $\mathbf{J}$ [19], which is defined by

$$\begin{bmatrix} \frac{\partial N}{\partial \xi} \\ \frac{\partial N}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial N}{\partial x} \\ \frac{\partial N}{\partial y} \end{bmatrix} = \mathbf{J} \cdot \begin{bmatrix} \frac{\partial N}{\partial x} \\ \frac{\partial N}{\partial y} \end{bmatrix} \ . \tag{3.28}$$

Now the numerical integration procedure known as Gaussian quadrature is applied, which approximates a definite integral with a weighted sum over a finite set of points [19]:

$$\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta)\, |\mathbf{J}|\, d\xi d\eta \cong \sum_{i=1}^{n_{gp}}\sum_{j=1}^{n_{gp}} W_i W_j f(\xi_i,\eta_j)\, |\mathbf{J}| \ . \tag{3.29}$$

W_i are the weight factors, ξ_i and η_i are the Gauss points and n_{gp} is the number of Gauss integration points, which specify that a polynomial of degree $2n_{gp}-1$ is integrated exactly by the method of Gauss-Legendre [19].

This method, which is the most common form of Gaussian quadrature, is chosen for the integration of the system matrices $[C]$, $[K]$ and $[R]$, with the exception of the velocity containing term

$$-\int_V ([B]^T)\rho c \mathbf{u}\, [N]\ dV \tag{3.30}$$

in the system matrix $[K]$, where a three-point Gauss-Lobatto quadrature rule is applied. In this rule, two of the three Gauss points lie on the ends of the integration interval. This was found to be more accurate due to the natural inclusion of boundary conditions. This was not further studied in a mathematically rigorous way, but rather, observed through testing. Because of the greater computational expense, this was programmed in such a way, that the term (3.30) is only evaluated for $\mathbf{u} \neq 0$. Gauss points and weight factors of the mentioned integration rules are given in Table 3.3.

Table 3.3: Gauss points and weight factors of the used quadrature rules [35]

Type of Gauss quadrature	Gauss points	Weight factors
Gauss-Legendre, $n_{gp} = 2$	$\pm 1/\sqrt{3}$	1
Gauss-Lobatto, $n_{gp} = 3$	0	4/3
	± 1	1/3

3.3 Convection modelling

As previously mentioned, the Navier-Stokes equation (3.4) is treated separately from the heat transfer part of the model via a finite difference discretization. In the early stages of this diploma work, an attempt was made to also discretize the Navier-Stokes equation with the finite element method, similar to the published work [11]. However, this method, using a penalty formulation for the pressure, could not be successfully implemented in the MATLAB code, as it did not lead to convergent solutions. To avoid this problem, an open source code, available on the course homepage (http://math.mit.edu/~gs/cse/) of 'Computational Science and Engineering' at the Massachusetts Institute of Technology, was implemented and extended. The documentation of this code can be found in [36]. In the following sections, the basic functioning of the existing MIT code will be described and later on, the modifications that were made in order to fit the here described problem will be explained.

At this point, it is necessary to point out that the methods in the following sections are merely a summary of the mentioned documentation [36] and not the author's own work, unless declared otherwise.

3.3.1 Two-dimensional Navier-Stokes equations

In two space dimensions, we derive for the nonlinear convective acceleration

$$(\mathbf{u}\cdot\nabla)\mathbf{u} = \left[\begin{pmatrix} u \\ v \end{pmatrix} \cdot \begin{pmatrix} \partial/\partial x \\ \partial/\partial y \end{pmatrix}\right] \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \dfrac{\partial u^2}{\partial x} + \dfrac{\partial uv}{\partial y} \\ \dfrac{\partial uv}{\partial x} + \dfrac{\partial v^2}{\partial y} \end{pmatrix} \tag{3.31}$$

and the term $\nabla(\nabla\cdot\mathbf{u})$ can be written as

$$\nabla(\nabla\cdot\mathbf{u}) = \Delta\mathbf{u} = \begin{pmatrix} \dfrac{\partial u}{\partial x^2} + \dfrac{\partial u}{\partial y^2} \\ \dfrac{\partial v}{\partial x^2} + \dfrac{\partial v}{\partial y^2} \end{pmatrix} . \tag{3.32}$$

The Navier-Stokes equation (3.4) and continuity equation in two space dimensions thus take the form

$$\frac{\partial u}{\partial t} + \frac{1}{\rho_0}\frac{\partial p}{\partial x} = -\frac{\partial u^2}{\partial x} - \frac{\partial uv}{\partial y} + \frac{\mu}{\rho_0}\left(\frac{\partial u}{\partial x^2} + \frac{\partial u}{\partial y^2}\right) + \frac{f_x}{\rho_0} \text{ and} \tag{3.33}$$

$$\frac{\partial v}{\partial t} + \frac{1}{\rho_0}\frac{\partial p}{\partial y} = -\frac{\partial v^2}{\partial y} - \frac{\partial uv}{\partial x} + \frac{\mu}{\rho_0}\left(\frac{\partial v}{\partial x^2} + \frac{\partial v}{\partial y^2}\right) + \frac{f_y}{\rho_0} , \tag{3.34}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = -\frac{\partial \rho}{\partial t} = 0 , \tag{3.35}$$

whereby f_x and f_y are the components of the buoyancy force (3.3).

3.3.2 Numerical solution approach

The numerical approach to solving the time dependent problem is the fractional step method [37], which splits the Navier-Stokes equations into equations that are significantly simpler to work with. The solution is then found by following a three step approach, which is outlined in the following, where $\mathbf{u}_t$ denotes the solution at the time t and $\mathbf{u}_{t+1}$ the final solution at the time $t+1$, while $\mathbf{u}^*$ and $\mathbf{u}^{**}$ denote intermediate solutions of the respective split equations.

1. **Explicit time step for nonlinear terms**

   The nonlinear convective acceleration term (3.31) is treated explicitly, which circumvents the implicit solution of a large nonlinear system of equations.

$$u^* - u_t - \Delta t \left(\frac{\partial u_t^2}{\partial x} + \frac{\partial u_t v_t}{\partial y} \right) = 0 \tag{3.36}$$

$$v^* - v_t - \Delta t \left(\frac{\partial u_t v_t}{\partial x} + \frac{\partial u_t^2}{\partial y} \right) = 0 \tag{3.37}$$

   This, however, introduces a Courant-Friedrichs-Lewy (CFL) condition, which limits the time step by a constant times the spatial resolution [38]. For a first-order upwind scheme, this CFL condition requires that, for stability, the conditions

$$CFL_{(x)} \equiv \frac{|u|_{max}\, \Delta t}{\Delta x} \leq 1 \tag{3.38}$$

   and

$$CFL_{(y)} \equiv \frac{|v|_{max}\, \Delta t}{\Delta y} \leq 1 \tag{3.39}$$

   hold [39].

2. **Implicit time step for viscosity terms**

   If the viscosity terms (3.32) were treated explicitly, this would result in a time step restriction proportional to the spatial discretization squared. This part of the equation is therefore handled by an implicit step, at the expense of two linear systems of equations to be solved in each time step:

$$u^{**} - u^* = \Delta t \frac{\mu}{\rho_0} \left(\frac{\partial u_{**}}{\partial x^2} + \frac{\partial u_{**}}{\partial y^2} \right) + \Delta t \frac{f_x}{\rho_0} \text{ and} \tag{3.40}$$

$$v^{**} - v^* = \Delta t \frac{\mu}{\rho_0} \left(\frac{\partial v_{**}}{\partial x^2} + \frac{\partial v_{**}}{\partial y^2} \right) + \Delta t \frac{f_y}{\rho_0} \ . \tag{3.41}$$

3. **Pressure correction step**

   The intermediate velocity field (u_{**}, v_{**}) is corrected by the gradient of pressure p_{t+1} to ensure incompressibility.

$$u_{t+1} - u^{**} = -\frac{\Delta t}{\rho_0} \frac{\partial p_{t+1}}{\partial x} \tag{3.42}$$

$$v_{t+1} - v^{**} = -\frac{\Delta t}{\rho_0} \frac{\partial p_{t+1}}{\partial y} \tag{3.43}$$

The pressure p_{t+1} is only given implicitly and can be obtained by solving a linear system of equations. Applying the divergence operator ∇ on both sides of equations (3.42) and (3.43), and taking into account the continuity equation $\nabla \mathbf{u}_{t+1} = 0$ leads to:

$$\Delta p_{t+1} = \frac{\rho_0}{\Delta t} \nabla \mathbf{u}^{**} \tag{3.44}$$

The pressure correction step can therefore be implemented as follows:

(a) Compute the gradient $\nabla \mathbf{u}^{**}$.

(b) Solve the Poisson equation $\Delta p_{t+1} = \frac{\rho_0}{\Delta t} \nabla \mathbf{u}^{**}$ for p_{t+1}.

(c) Compute the pressure gradient ∇p_{t+1}.

(d) Update the velocity field $\mathbf{u}_{t+1} = \mathbf{u}^{**} - \frac{\Delta t}{\rho_0} \nabla p_{t+1}$.

When solving the Poisson equation, the question arises on the appropriate choice of boundary conditions for the pressure p. In the documentation [36], it is stated that a standard approach is to prescribe homogenous Neumann boundary conditions wherever no-slip boundary conditions are prescribed to the velocity field, which, in this case is on every boundary. This implies that the pressure is only defined up to a constant. However, absolute pressure information is not needed because only the gradient of p is needed in the pressure correction step.

It can be noted that combining equations (3.36), (3.40) and (3.42) again yields the full Navier-Stokes equation system (3.33). The same is of course true for the velocity component v in equation (3.34).

3.3.3 Spatial discretization

The spatial discretization of the Navier-Stokes equation is done via the finite difference method, as introduced in Section 2.3.2. A staggered grid (see Figure 3.2) based on the finite element grid is used to solve the heat transfer problem, where the pressure

p is defined in the element centers, u is placed in the middle of the vertical element borders and v is placed in the middle of the horizontal element borders.

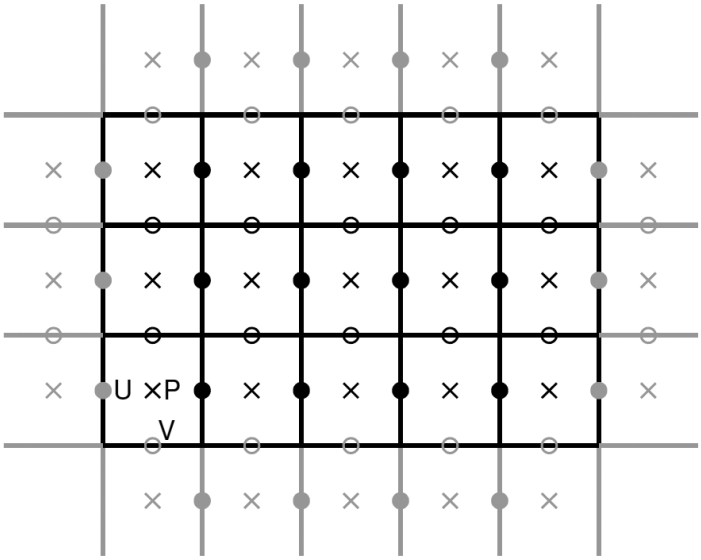


Figure 3.2: Two-dimensional domain discretized into a staggered grid [36]

The second and first derivatives are approximated by centered finite difference stencils, as illustrated here for $\Delta u_{i,j}$ and $\left(\frac{\partial u}{\partial x}\right)_{i,j}$ at the grid point (i, j):

$$\Delta u_{i,j} = \left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j} + \left(\frac{\partial^2 u}{\partial y^2}\right)_{i,j} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta_x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta_y^2} \tag{3.45}$$

$$\left(\frac{\partial u}{\partial x}\right)_{i,j} \approx \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} \ . \tag{3.46}$$

This approximation for the first derivative can yield instabilities [36], but the centered stencil

$$\left(\frac{\partial u}{\partial x}\right)_{i+\frac{1}{2},j} \approx \frac{u_{i+1,j} - u_{i,j}}{\Delta x} \tag{3.47}$$

is a stable approximation for $\dfrac{\partial u}{\partial x}$ between two grid points [36]. This aids the staggered grid method, where this position is exactly the position of the pressure $p_{i,j}$.

For the nonlinear terms, a combination of the centered differencing and upwinding approach is implemented, with a transition parameter $\gamma \in [0,1]$ defined as

$$\gamma = min\left(1.2 \cdot \Delta t \cdot max\left(\underset{i,j}{max}\left|u_{i,j}\right|, \underset{i,j}{max}\left|v_{i,j}\right|\right), 1\right) \ . \tag{3.48}$$

The value of γ can be interpreted as the maximum fraction of a cell whose information can travel in one time step, multiplied by 1.2, which is an empirical factor that brings the approximation scheme more towards upwinding and is, from experience, more accurate [40].

With the introduction of averaged quantities

$$\bar{u}^h_{i+\frac{1}{2},j} = \frac{u_{i+1,j} + u_{i,j}}{2} \text{ and} \tag{3.49}$$

$$\bar{u}^v_{i,j+\frac{1}{2}} = \frac{u_{i,j+1} + u_{i,j}}{2} \ , \tag{3.50}$$

(equally for the component v) using the superscript h and v to indicate a horizontal or vertical averaged quantity and denoting the differenced quantities likewise with $\tilde{u}^h, \tilde{u}^v$, etc., the nonlinear terms in equations (3.36) and (3.37) can be written as the linear combinations

$$\left(\frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y}\right) = \frac{\partial}{\partial x}\left(\left(\bar{u}^h\right)^2 - \gamma\left|\bar{u}^h\right|\tilde{u}^h\right) + \frac{\partial}{\partial y}\left(\left(\bar{u}^v\bar{v}^h\right) - \gamma\left|\bar{v}^h\right|\tilde{u}^v\right) \text{ and} \tag{3.51}$$

$$\left(\frac{\partial uv}{\partial x} + \frac{\partial u^2}{\partial y}\right) = \frac{\partial}{\partial x}\left(\left(\bar{u}^v\bar{v}^h\right) - \gamma\left|\bar{u}^v\right|\tilde{v}^h\right) + \frac{\partial}{\partial y}\left(\left(\bar{v}^v\right)^2 - \gamma\left|\bar{v}^v\right|\tilde{v}^v\right) \ , \tag{3.52}$$

which are again approximated by centered finite difference stencils.

3.3.4 Boundary conditions

The boundary conditions prescribed for the model were already stated in Section 3.1.4, but since a staggered grid was introduced, applying the correct boundary conditions requires care, as some grid points lie on a boundary, while others have a boundary between them. For the no-slip boundary conditions in place, we can obtain the condition

$$u_{i,j} = 0, \text{ if } i = 1 \text{ and } n_x + 1 \tag{3.53}$$

at the west and east domain borders and

$$v_{i,j} = 0, \text{ if } j = 1 \text{ and } n_y + 1 \tag{3.54}$$

at the south and north borders. To force zero velocity at the boundary, the conditions

$$\begin{aligned} u_{i,j=1} &= -\, u_{i,j=2} \quad \text{and} \\ u_{i,j=n_y+1} &= -\, u_{i,j=n_y+2} \end{aligned} \tag{3.55}$$

at the south and north domain border and

$$\begin{aligned} v_{i=1,j} &= -\, v_{i=2,j} \quad \text{and} \\ v_{i=n_x+1,j} &= -\, v_{i=n_x+2,j} \end{aligned} \tag{3.56}$$

at the west and east border are also prescribed. The homogenous Neumann boundary conditions for the pressure p yield the following conditions at the four boundaries:

$$\begin{aligned} p_{i=1,j} &= -\, p_{i=2,j} \\ p_{i=n_x+1,j} &= -\, p_{i=n_x+2,j} \\ p_{i,j=1} &= -\, p_{i,j=2} \\ p_{i,j=n_y+1} &= -\, p_{i,j=n_y+2} \ . \end{aligned} \tag{3.57}$$

3.3.5 Adaptations to the existing MATLAB code

Although the basic structure, which was explained in Section 3.3 and documented in [36], was used as foundation for the implemented code, much of it had to be modified in order to adapt it the problem described here.

- **External force term**

  As the existing code did not include outer force terms, the Boussinesq force term was added to the code and treated implicitly in equations (3.40) and (3.41).

- **Solid phase modelling**

  To model the liquid and solid phases in the domain, another part of the force term in the implicit equations mentioned above, was added. This can be seen as an external force density, which inhibits fluid motion in the solid part [41]:

$$\mathbf{f} = -\frac{(1-a)^2}{a^3 + 0.001} 10^8 \cdot \mathbf{u}^{**} \tag{3.58}$$

The large number 10^8 was chosen to force the velocity to zero via the damping term. The liquid fraction parameter is $a = 0$ in the solid phase and rises over the mushy region to $a = 1$ in the liquid phase.

- **Phase boundaries**

  The boundary conditions defined in 3.3.4 are in place at the domain boundaries and the phase boundaries. To ensure this, the existing velocity field is modified to suit the conditions before the explicit time step is performed.

  For the pressure p, this step is more complex, as the boundary conditions are implemented in the Laplace-operator to solve the Poisson equation (3.44). Therefore, the Laplace-operator must be assembled anew in every time step.

Chapter 4

MATLAB implementation

This chapter deals with the final form of the MATLAB implementation of the numerical model described in Chapter 3. The basic structure of the code is outlined in Section 4.1, while the main functions are explained in some detail in Section 4.2. The code of these important code parts is also listed in the appendix A and referred to in this chapter.

For those interested in applying the code to conduct simulations on their own, it is highly recommended to look up Section 4.3. Therein, a detailed guide on how to specify input data for simulations is given.

It should be noted that the herein documented code was created over a period of several months. Many extensions to the initial version have been made, and the code was significantly modified several times. Although the code was written in a very comprehensible manner in the early development, measures to reduce the computational effort, such as preprocessing, vectorization and the usage of the sparse matrix format, made it much more complex and probably quite difficult to read for unfamiliar users. However, it should be stressed that readability was not of crucial importance for this implementation, since this code was not developed for educational purposes, but for a designated application.

For the sake of better readability, filenames, the names of MATLAB functions or scripts and variable names are denoted by a different font: `name`.

4.1 Structure of the program

The MATLAB program developed was built as one main function with numerous sub-functions. The execution procedure is represented by the flowchart in Figure

4.1. The simulation program is executed by calling the main function PCMsolver and passing the path to the input datafile, which specifies all simulation parameters as input argument. This is discussed in detail in Section 4.3.

Preprocessing

In preprocessing, input data is loaded to the workspace where all important parameters are saved in the structure variable param. The finite element mesh is created by calling the function mesh2d (see 4.2.1). All data arrays are initialized, most notably, the initial temperature array T_old and velocity arrays U_old and V_old. The function shape_mat preprocesses the shape function values at the Gauss integration points. This is useful here, because all elements are of the same size and therefore the values do not change. The calculated values are stored in the structure variable gauss.

The function FEM_preprocessor adds further information to this structured variable. It evaluates the values of the density, specific heat capacity and heat conductivity for every Fauss point of every element in the domain. Although the specific heat capacity must be recalculated at every time step for the PCM material, this step saves work on the time iteration. Furthermore, boundary informations are preprocessed and the relations for the sparse matrix assembly, which is carried out by the function heat2Delem_new, are defined.

Time stepping

The time stepping is done in a for-loop, calling firstly the function heat2Delem_new, which calculates the finite element matrices $[C]$, $[K]$ and $[R]$, defined by the equations (3.24), (3.25) and (3.26). The equation system (3.23) is then solved by the function solve_euler_back for the nodal temperature values at the next time step T_new.

If the simulation is run without considering convection, the body of the for-loop ends after saving the simulation results of the time step and reinitializing the arrays of variables T_old = T_new, U_old = U_new and V_old = V_new for the next time step. Otherwise, the velocities are updated as will be discussed below.

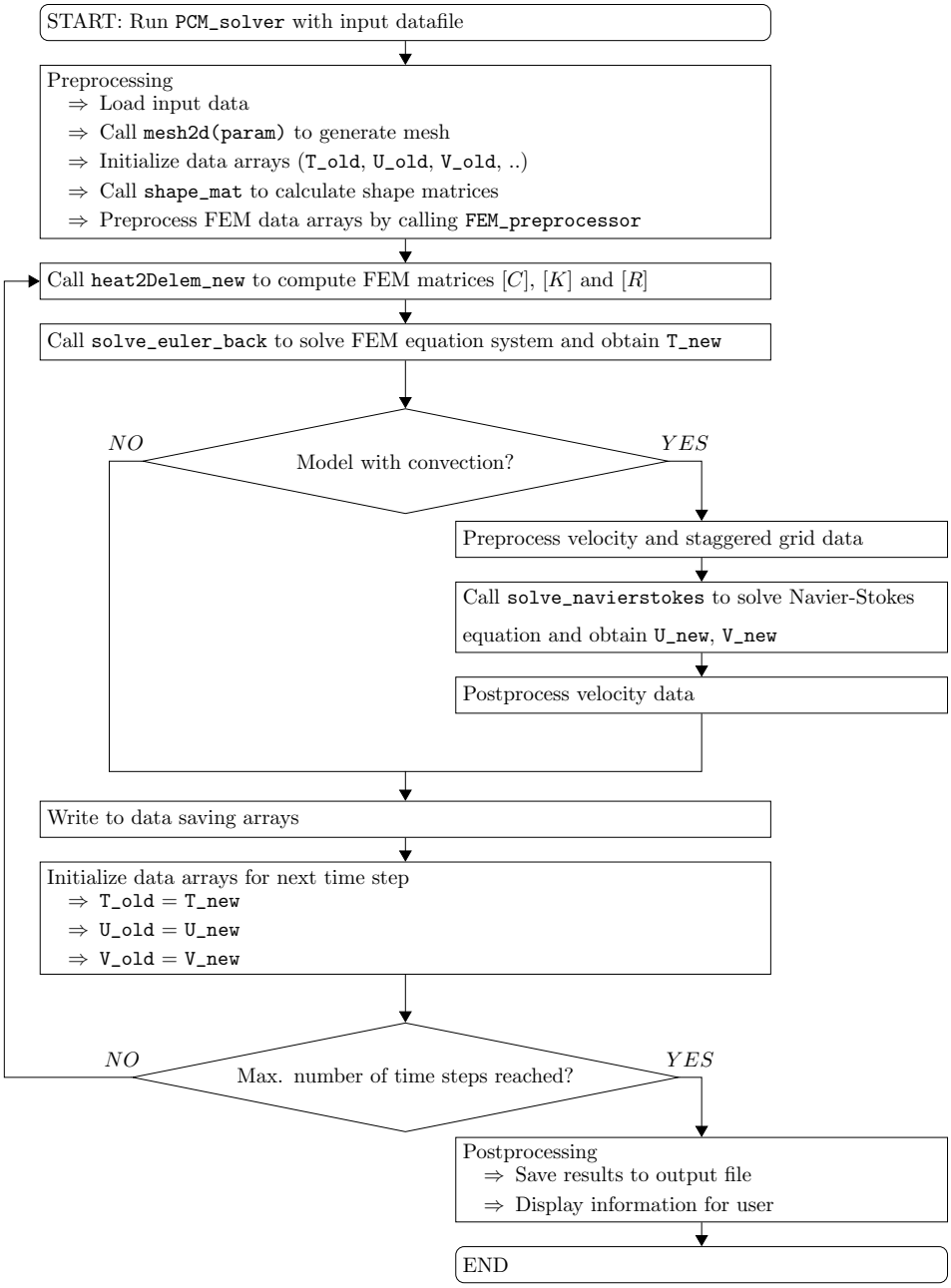


Figure 4.1: Flowchart diagram of FEM model PCMsolver

Note. Code given in A.1

Convection calculations

The velocity is updated in the function solve_navierstokes, which essentially solves the Navier-Stokes equations on a staggered grid by the finite difference scheme explained in Section 3.3. Although these updated velocity values are, at this point, close to zero in solid domain areas, because of the force term introduced in the implicit viscosity step (3.58), postprocessing must be performed. Therein, all velocity components, enclosed in a completely solid finite element, are set to zero. The no-slip boundary values are then newly prescribed to the updated velocities as stated by equations (3.55) and (3.56). These steps are performed for the domain boundaries and also for the liquid-solid phase boundaries, which can be observed in detail in lines 127 to 146 of listing A.1.

Postprocessing

After the last time step is performed, the complete simulated data is saved to the results file and optional user information is displayed.

4.2 Description of the key functions

In this section, the most substantial sub-functions of the main program are explained. This is done by highlighting the basic principles of implementing the model in MATLAB, as described in Chapter 3. The main function PCMsolver was already discussed in detail in the previous section, its listing A.1 can be found in the appendix A, together with the other key functions.

4.2.1 Mesh generation in mesh2d

The finite element mesh is generated in the function mesh2d (A.2) by the call

```
[param] = mesh2d(param);
```

in PCMsolver. As already mentioned in Section 3.2.1, the two-dimensional domain is split into an equidistant mesh of $n_{el} = n_x \cdot n_y$ four-noded bilinear rectangular finite elements, which yields the total sum of $n_n = (n_x + 1) \cdot (n_y + 1)$ nodes. These elements and nodes are arranged in the domain from left to right and from bottom to top. This is best understood by looking at Figure 4.2, which shows an example of the element and node arrangement for a mesh with $n_x = 2$ and $n_y = 2$. The information about the connectivity between the nodes and the elements is saved to the array param.mesh.IEN.

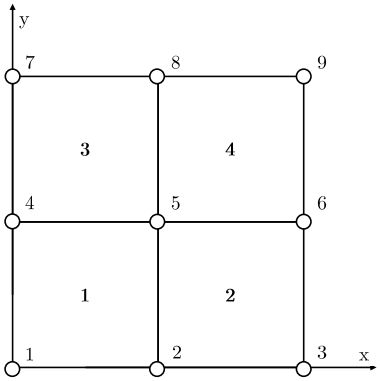


Figure 4.2: Image of a 2x2 mesh, with numbered nodes and elements

Furthermore, this function not only generates the coordinates of element nodes, but also those of the staggered grid points. Additionally, boundary elements are flagged, which allows quick identification when calculating the heat flux at the boundaries, and the array param.mesh.elem_mat is created, which contains indexing information about the element materials based on the geometry specifications.

4.2.2 Preprocessing operations

The main purpose of the functions shape_mat (A.3) and FEM_preprocessor (A.6) was already highlighted in 4.1. These are executed once in the simulation by the call:

```
[gauss] = shape_mat(param);
```

```
[gauss] = FEM_preprocessor(T_old,gauss,param);
```

In shape_mat, all occurring terms of shape matrices $[N]$ and $[B]$ in the system matrices (3.24), (3.25) and (3.26) are evaluated for each Gauss integration point, as well as the determinant of the Jacobian matrix (3.28). This very useful preprocessing method is possible since, in this implementation, all elements are the same size and do not change during the simulation. The shape functions are calculated by the sub-functions NmatHeat2D and BmatHeat2D, which essentially contain the formulas (3.17) and (3.15).

Function FEM_preprocessor calculates further information for the FEM matrix assembly. It evaluates the values of the density, specific heat capacity and heat conductivity for every Gauss point of every element in the domain and preprocesses boundary informations, as already explained in 4.1. In addition, sparse matrix assembly relations are defined, to be used later in heat2Delem_new. The sparse matrix format in MATLAB provides efficient storage of data and speeds up the processing of that data. A good example of how the element matrices are assembled to the global matrix can be found in [42], for example.

4.2.3 FEM Matrix calculation in heat2Delemnew

The main FEM processing is done in heat2Delem_new (A.7), where the matrices $[C]$, $[K]$ and $[R]$, defined by the equations (3.24), (3.25), (3.26) are evaluated by an implementation of these equations. The function is called in PCMsolver by:

```
[C,K,R,Qin_e,Qout_e,H(t),H_l(t)] = ...
heat2Delem_new(t,T_old,Uwb,Vwb,p_el,gauss,param);
```

The heat conductivity and specific (apparent) heat capacities are evaluated at each Gauss point, which is only done for elements containing PCM material. For the remaining elements, the preprocessed data of FEM_preprocessor is used, which also included the preprocessed shape matrix terms. The global matrices are then assembled using the relations defined in the aforementioned function. The apparent heat capacity

is calculated by the function c_var (A.8), which implements definition (2.20) to account for phase change in the PCM material.

Regarding the convective term in matrix (3.25), the integration is performed by the Gauss-Lobatto quadrature instead of the common Gauss-Legendre quadrature, as already explained in Section 3.2.3. The arrays containing the velocity values u and v are in line with the scheme of the staggered grid (see Section 3.3) with already implemented boundary conditions. These values are used to interpolate the values at the Gauss points.

4.2.4 Solving the matrix equations

The function solve_euler_back (A.10) is called thus:

```
T_new = solve_euler_back(T_old,C,K,R,param);
```

It contains only one definition:

```
Tnew = Told + param.mesh.d_t*...
(  (C + param.mesh.d_t*K)\(R-K*Told)  );
```

This is the MATLAB implementation of the applied backwards Euler step, yielding the formula (2.38) for the updated temperature value. The sparse matrix format allows very efficient evaluation of this step.

4.2.5 Solving the Navier Stokes equations

The convection modelling discussed in detail in Section 3.3 is performed in the function solve_navierstokes (A.12), which is called in the main function PCMsolver thus:

```
[U_new,V_new] = solve_navierstokes(U_old,V_old,T_ij,p_el,param
    );
```

Therein, the force terms, which inhibit fluid motion in the solid elements, are pre-processed, as are the temperatures that are needed for the Boussinesq approximation terms. The remaining function consists primarily of an implementation of the code documented in [36].

Worth mentioning is the function Laplace_p (A.13). It was developed in order to compute a finite difference Laplace stencil for the pressure p which is consistent with homogeneous Neumann boundary conditions on the domain boundaries, as well as the internal phase boundaries.

4.3 Usage information

The main function PCMsolver must be run by passing a string as an input argument. This string contains the path to the MATLAB file containing the simulation input data. For instance, if the file is called input_datafile.m the simulation is started by calling:

```
PCMsolver('input_datafile');
```

All relevant simulation input data can be specified using such an input datafile. The code Listing 4.1 gives an example of a typical declaration structure, which is recommended. This file actually represents a typical input datafile used in the simulations carried out for the parameter study discussed in Section 6.2. In the following, both essential and optional input parameters are discussed in detail, referring to the respective lines in code Listing 4.1.

Code Listing 4.1: Listing of a typical input datafile input_datafile.m

```
%% Input Data File
% The material and computational parameters are specified and written to
% the structure array param.

%% name for file saving
param.save.name = 'results';

%% simulate with convection
param.convection = 1;   %(0 = off,  1 = on)

%% geometry specifications
% length of material domains in x-direction (left to right)
param.geom.x_length_vec = [0.002 0.118];
% length of material domains in y-direction (bottom to top)
param.geom.y_length_vec = [0.001 0.023 0.003];%
% structure of simulated domain (left to right and top to bottom)
param.geom.mat_mtrx = flipud([3 3
                              3 1;
                              3 3]);
% definition of material specifier
% must be in accordance with definition of material properties below
% PCM material specifier must be 1
param.geom.material = {1,'pcm'; % PCM material
                       2,'stl'; % steel
                       3,'alu'; % aluminium
                       4,'air'};% air

```

```
param.geom.x_length = sum(param.geom.x_length_vec); % length of PCM in x-
    direction
param.geom.y_length = sum(param.geom.y_length_vec); % length of PCM in y-
    direction

%% mesh specifications
nx_min = param.geom.x_length/0.0005;    % minimum number of elements in x
    direction
ny_min = param.geom.y_length/0.0005;    % minimum number of elements in y
    direction
nx = calc_grid(nx_min,param.geom.x_length_vec);
ny = calc_grid(ny_min,param.geom.y_length_vec);
param.mesh.nx   = nx;
param.mesh.ny   = ny;
param.mesh.nel  = nx*ny;                % number of elements
param.mesh.non  = (nx+1)*(ny+1);        % number of nodes
param.mesh.ngp  = 2;                    % number of gauss integration points
param.geom.dx = param.geom.x_length/nx;
param.geom.dy = param.geom.y_length/ny;

%% time step specifications
param.mesh.tt = 3*60*60;                                % total time
param.mesh.d_t = 0.05;                                  % time step
param.mesh.not = ceil(param.mesh.tt/param.mesh.d_t);    % number of time steps

%% initial and boundary condition
% T_init [°C] = initial temperature for element nodes
param.icbc.T_init=218*ones(param.mesh.non,1);
% T_in [°C] = input temperature dependent of time
param.icbc.Tin = @(t,y) 230; %(t<=param.mesh.not)*200+(t>param.mesh.not)*100;
% alpha_in [W / m*K] = heat conductivity dependent of y and time
%yh = param.geom.y_length/2;
param.icbc.alpha_in = @(t,y) 700; %(y<yh)*100+(y>=yh)*20;
% T_out [°C] = output temperature dependent of time
param.icbc.Tout = @(t,y) 20;      % output temperature [°C]
% alpha_out [W / m*K] = heat conductivity to environment
param.icbc.alpha_out = @(t,y) 1e-2;    % heat conductivity [W / m*K]

%% material properties
% material properties of pcm
param.pcm.rho = 2050;           % density                   [kg / m^3] (solid
    )
param.pcm.Tm = 220;             % melting temperature       [°C]
param.pcm.dTm = 1.0;            % mushy region size         [°C]
param.pcm.c_s = 1350;           % heat capacity of solid    [J / kg*K]
param.pcm.c_l = 1492;           % heat capacity of liquid   [J / kg*K]
param.pcm.lh = 108*1000;        % latent heat of PCM        [J / kg]
param.pcm.k_s = 0.457;          % heat conductivity, solid  [W / m*K]
param.pcm.k_l = 0.435;          % heat conductivity, liquid [W / m*K]
% convective properties of pcm
param.pcm.beta = 3.5e-4;        % thermal expansion coeff.  [1 / K]
param.pcm.grav = 9.81;          % standard gravity          [m / s^2]
param.pcm.mue = 5.8e-3;         % viscosity                 [m^2 / s]
param.pcm.phi = 0;          % angle of gravitation vector to y-axis
% examples of angle phi:
% phi = 0 (default option)  -> gravitation in -y direction
% phi = pi                  -> gravitation in +y direction
% phi = pi/2                -> gravitation in -x direction
% phi = -pi/2               -> gravitation in +x direction

% material properties of aluminium
param.alu.rho = 2700;           % density                   [kg / m^3] (solid
    )
param.alu.Tm = 1000;            % melting temperature       [°C]
param.alu.dTm = 0;              % area of melting           [°C]
param.alu.c_s = 910;            % heat capacity of solid    [J / kg*K]
param.alu.lh = 0;               % latent heat of PCM        [J / kg]
param.alu.k_s = 237;            % heat conductivity, solid  [W / m*K]

% material properties of steel
param.stl.rho = 7800;           % density                   [kg / m^3] (solid
    )
param.stl.Tm = 1000;            % melting temperature       [°C]
param.stl.dTm = 0;              % area of melting           [°C]
param.stl.c_s = 540;            % heat capacity of solid    [J / kg*K]
param.stl.lh = 0;               % latent heat of PCM        [J / kg]
param.stl.k_s = 51;         % heat conductivity, solid  [W / m*K]

% material properties of air
param.air.rho = 1;          % density                   [kg / m^3] (solid)
```

```
101 param.air.Tm = 1000;            % melting temperature       [°C]
102 param.air.dTm = 0;              % area of melting           [°C]
103 param.air.c_s = 1000;           % heat capacity of solid    [J / kg*K]
104 param.air.lh = 0;               % latent heat of PCM        [J / kg]
105 param.air.k_s = 0.024;      % heat conductivity, solid  [W / m*K]
106
107 %% data recovery
108 param.save.intervall = 6*ceil(1/param.mesh.d_t);        % time step intervall
109                                                      % for T,U,V to be saved
110 param.save.make_snapshot = 1;  %(0 = off,  1 = on)  % dump snapshots of data
111 param.save.snapshot_int = 120;  % time step intervall to make snapshots
112
113 %% display information for user
114 % (0 = off,  1 = on)
115 param.display.waitbar = 1;  % show waitbar during run
116 % info: each waitbar update needs about 0.2 seconds computation time
117 param.display.update = 10;  % minimum time between waitbar updates in seconds
118 param.display.Tplot = 0;    % plot temperature distribution at the end
119 param.display.Vplot = 0;    % plot velocity distribution at the end
120 param.display.matplot = 0;  % plot visualisation of material in domain
```

As mentioned above, all input parameters are specified in the structure array param. For more clarity, the fields of param again contain structure arrays.

In line 6, the basic string for the filenames of the result files is declared. Line 9 contains binary information about whether the effect of natural convection is to be considered in this simulation.

Geometry specifications

The geometry specifications are defined in lines 11 to 30. The length of different material domains is specified using vectors and the organisation can be easily understood by looking at Figure 6.1. The field param.geom.mat_mtrx contains a matrix of indices specifying certain materials. These indices are to be defined arbitrarily in param.geom.material, except for the fixed index 1, which refers to the PCM material. The syntax of the matrix formulated material declaration chosen here can again be best put into context by observing Figure 6.1.

Mesh specifications

The essential mesh specifications are defined in param.mesh.nx and param.mesh.ny (lines 36 and 37), which are the number of elements in x direction and y direction respectively. Lines 32 to 35 may be used by calling the created function calc_grid to

calculate a number of elements that is compatible with the material domains specified in lines 13 and 15, in the means that no element contains more than one material.

Time step specifications

Lines 45 and 46 contain the time step specifications, which are the total simulation time param.mesh.tt and the time step size param.mesh.d_t.

Initial and boundary conditions

The possible values for initial and boundary conditions are defined in lines 49 to 60, which are the heat conductivities α_{in} and α_{out} and temperatures T_{in} and T_{out} belonging to the Robin boundary conditions on the west and east side of the domain. The initial temperature field is prescribed for each element through the vector param.icbc.T_init.

Material properties

Material properties are defined in structure arrays and saved in fields of param, of which the names must be declared in accordance with the definition of material specifiers in param.geom.mat_mtrx. The basic thermophysical material properties for the phase change material are defined in lines 64 to 71, and so is also the mushy region size ϵ in line 66. The same can be done for other materials, as given here for aluminium, steel and air in lines 84 to 105. The convective simulation parameters are defined in lines 73 to 76 and contain the thermal expansion coefficient param.pcm.beta, the viscosity param.pcm.mue and the angle param.pcm.phi. This angle, which is defined as the angle of the gravitational vector to the y-axis, was introduced for the parameter study in Section 6.2 to study different domain orientations. For the intuitive case that the y-axis is aligned to the gravitational force vector, this angle takes the value zero. Other values can be defined in radian units in a clockwise manner, as given in the example values in lines 78 to 81.

Data recovery

Some options are available regarding the recovery of the simulation data of the node temperature values and velocity value matrices. param.save.intervall defines the number of time steps between moments at which the mentioned quantities are to be saved. This is useful to keep the amount of aggregated data moderate. Snapshots of the data can be saved by setting the binary variable param.save.make_snapshot to the value 1, at intervals defined by the associated variable param.save.snapshot_int. This option allows insights into the results of ongoing simulations.

Display parameters

With the binary variable param.display.waitbar, the display of a waitbar, showing the progress of the simulation and remaining time until completion, can be switched on or off. The corresponding variable param.display.update declares the time between waitbar updates. However, this option comes with the disadvantage of significant computational effort for small update intervals, because of the rather slow MATLAB-internal function draw. More visualization options are available in lines 118 to 120, which can be enabled by setting them to the value 1 and disabled by setting them to the value 0.

Chapter 5

Validation

In this chapter, the results of the model validation for the latent heat storage model are presented. The validation was conducted in three steps.

The first validation step was to compare the results of a simulation where only heat conduction was considered to those of the analytical solution of the Stefan problem, introduced in Section 2.1.4.

The second test case is the classic problem described in [5], that is phase change in a cavity filled with gallium with buoyancy forces. This test case aimed at validating the convection model described in Section 3.3.

In the next step, a cross-validation with the CFD software ANSYS Fluent was performed. As part of the parameter study, which will be discussed in Chapter 6, the model results are compared to those replicated with the Fluent software package.

Another test case was performed which was lass a validation attempt than a test of the model stability and feasibility in certain simulation scenarios. This test scenario alternated between charging and discharging a PCM storage to see if numerical problems arose and if the results complied with physical principles.

For the two validation attempts of the heat conduction model and convection model, a mesh convergence analysis was performed to study the effect of the discretization parameters mesh size and time step on the numerical solution. Also, the effect of the chosen mushy region temperature range and possible complications are scrutinized, as are the effect of the aspect ratio of the chosen finite elements.

5.1 Validation of heat conduction model

To validate the accuracy and convergence of the model without convection, the one-dimensional Stefan problem was considered, as described in Section 2.1.4. The material parameters used in this simulation are those of KNO_3-$NaNO_3$ given in Table 3.1, but to simplify the analytical solution, the specific heat capacity $c_l = c_s$ and thermal conductivity $k_l = k_s$ were set to the values of the solid substance. The melting point was defined as $T_m = 220\,°C$ and the left wall input temperature as $235\,°C$. The initial temperature of the simulated domain was naturally set to $T_{init} = T_m - \epsilon$, with the half width of the mushy region ϵ. The heat conduction was simulated in a domain with the length of $50\,mm$ and over a period of five hours, $5\,h$, in which the melting front could not quite spread over the whole domain.

This validation simulation was carried out for different values of the time step size Δt, mesh size Δx and mushy region parameter ϵ, where in each case the other two parameters were held at a constant value. Therefore, the influence of the mentioned numbers on the result was studied separately as given in the sections below.

The RMSE (root mean square error) statistical values [43]

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(s_{analytical} - s_{simulations}\right)^2} \tag{5.1}$$

were used to compare analytical predictions and simulation results, where s is the position of the melting front and n the number of comparison points. The melting front positions were evaluated and compared at $n = 1800$ points in time, spread equidistantly over the five hours of simulation time.

To quantify the convergence of the solution with finer mesh and smaller time steps, the melting front positions were also compared to the most precise simulation in that regard. For this purpose, the values $s_{analytical}$ in equation (5.1) are compared with those of the most precise simulation.

Effect of mesh refinement

To study the effect of mesh refinement, the Stefan problem was simulated for different values of the mesh size Δx, while the time step size $\Delta t = 1\,s$ and mushy region parameter $\epsilon = 1\,°C$ were held constant. The RMSE compared to the analytical prediction and the RMSE compared to the simulation with $\Delta x = 0.1\,mm$ are given in Table 5.1.

Table 5.1: RMSE of melting front position for different values of mesh size or number of elements compared to Stefan solution (analytical) and $\Delta x = 0.1\,mm$ simulation (convergence)

Mesh size $\frac{\Delta x}{mm}$	5	2	1	0.5	0.25	0.1
Number of elements	10	25	50	100	200	500
$\frac{\text{RMSE}}{10^{-4}\,mm}$ (analytical)	9.36	2.03	1.86	2.01	2.02	2.03
$\frac{\text{RMSE}}{10^{-4}\,mm}$ (convergence)	11.0	2.07	0.76	0.07	0.02	-

With the RMSE compared to the most precise simulation with $\Delta x = 0.1\,mm$, we can observe very fast convergence with decreasing mesh size. However, it can be noted that, compared to the analytical solution, the simulation does not become more accurate with decreasing mesh size after $\Delta x = 2\,mm$. Obviously, the accuracy limiting factor, in this case, lies in other simulation parameters.

Effect of applied time step

In the next step, the mesh size $\Delta x = 0.5\,mm$ and mushy region parameter $\epsilon = 1\,°C$ were held constant, while the simulation of the Stefan problem was carried out for different values of the time step size Δt. Again, the RMSE values are given in Table 5.2, in the same fashion as in Section 5.1.

Table 5.2: RMSE of melting front position for different values of time step size compared to Stefan solution (analytical) and $\Delta t = 0.1\,s$ simulation (convergence)

Time step size $\frac{\Delta t}{s}$	10	5	1	0.5	0.1
$\frac{\text{RMSE}}{10^{-4}\,mm}$ (analytical)	3.66	2.16	2.01	2.10	2.12
$\frac{\text{RMSE}}{10^{-4}\,mm}$ (convergence)	4.34	2.19	0.33	0.08	-

Again, the RMSE compared to the most precise simulation with $\Delta t = 0.1\,s$ indicates very fast convergence with decreasing time step size. The error compared to the analytical solution remains at a rather steady value for time steps $\Delta t = 5\,s$ and smaller. This, however, can be very different for other mushy region sizes ϵ. For smaller values of the latter, the time step size should also be decreased. Otherwise, the phase change is partly or completely skipped in certain finite elements, which is a known problem of the apparent heat capacity method and which will be clearer in the next subsection.

Effect of mushy region size

The effect of the mushy region size ϵ on the simulation result of the Stefan problem was examined under constant values of mesh size $\Delta x = 0.5\,mm$ and time step size $\Delta t = 1\,s$. The error values of the melting front position compared to the analytical solution are given in Table 5.3.

Table 5.3: RMSE of melting front position for different values of mushy region size ϵ compared to Stefan solution (analytical)

Mushy region size $\frac{\epsilon}{^\circ C}$	4	2	1	0.5	0.2	0.1
$\frac{\text{RMSE}}{10^{-4}\,mm}$ (analytical)	8.00	4.04	2.01	1.06	3.04	8.28

The quantities in Table 5.3 are best put into context by looking at the time evolution of the melting fronts of these different simulations, given in Figure 5.1. The melting front lags slightly behind the analytical solution for large mushy region size values, which can be explained by the distinctly altered actual physical problem, the Stefan problem, where no mushy region exists. Having said this, for small values of the mushy region size, the simulation does not become accurate, due to the nature of the apparent heat capacity method. As mentioned above, the phase change is partly or completely skipped in certain finite elements if the mushy region is small and the time step is large. This causes a higher RMSE value for the simulations with mushy region size $\epsilon = 0.2\,°C$ and $\epsilon = 0.1\,°C$, compared to the simulation with $\epsilon = 0.5\,°C$. In addition, the effect of this error can be seen very clearly for the simulated melting front with mushy region size $\epsilon = 0.1\,°C$, which is plotted as a dashed red line in Figure 5.1.

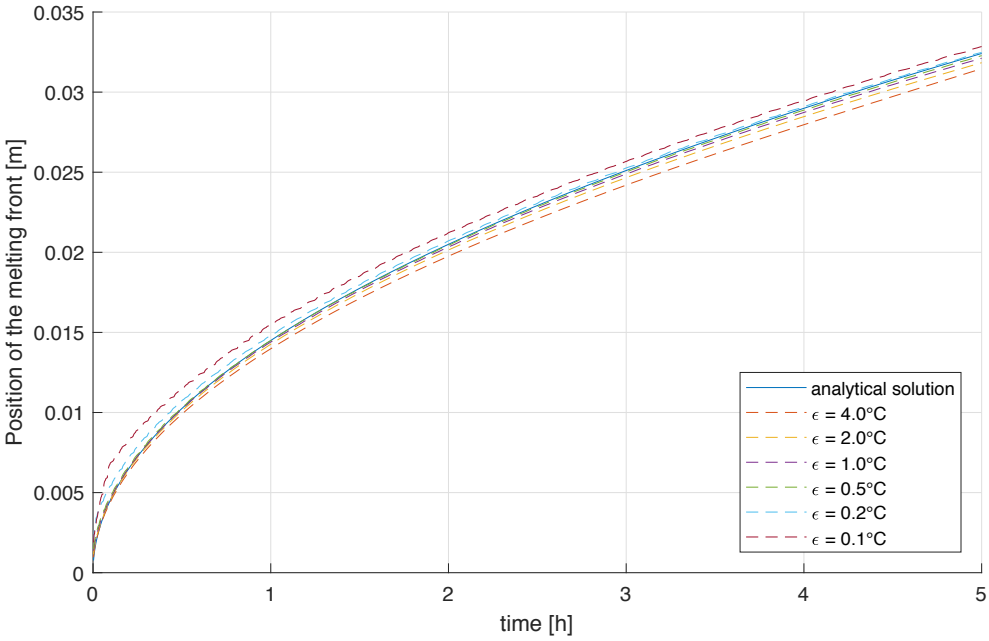


Figure 5.1: Time evolution of the melting front of simulations with different values of the mushy region size ϵ compared to the melting front of the analytical solution

Looking at Figure 5.1, we can obtain that the error values for the simulations with mushy region size $\epsilon = 1\,°C$ and $\epsilon = 0.5\,°C$ given in Table 5.3 should be taken into consideration with care. These small deviations from the analytical solution most probably arise from two competing sources of error, which cancel each other out. Firstly, phase change is partly skipped in some elements during the early stage of the simulation with high temperature gradient, which causes the melting front to be slightly more advanced, compared to the analytical solution. Secondly, the physical problem is distinctly altered by introducing a mushy region, which causes a slower progress of the melting front, which therefore cancels out the first mentioned error.

Based on the findings of the last section, it is suggested that the magnitudes of the time step and mushy region size be chosen so that the mushy region always overlaps its previous position, ensuring that no finite element skips the phase change. Due to the nature of the apparent heat capacity method, these elements would not be exposed to the effect of latent heat. These parameters should thus be chosen with care, and the findings of this section might help to give a basic idea of the necessary scaling.

Final 1D validation result

A final 1D simulation was carried out, aiming for an excellent prediction of the Stefan problem. For this, the mesh size was chosen as $\Delta x = 0.5\,mm$, which was already seen as a good approximation. The time step size $\Delta t = 0.02\,s$ was chosen very small to easily handle a small mushy region of $\epsilon = 0.1\,°C$.

The RMSE value compared to the analytical solution was found to be $1.08 \cdot 10^{-4}\,mm$ but the absolute deviation decreased with growing simulation time. Thus, the relative error of the melting front prediction was well under $1\,\%$, except for the early stages of the simulation. On the basis of the above, it can be concluded that the developed model gives an excellent approximation of heat conduction problems if the simulation parameters are carefully chosen.

5.2 Validation of convection model

5.2.1 Test case

The validation of the convection model is based on the report [5], in which the role of natural convection in solid-liquid-interface motion is studied during melting and solidification of the metal gallium on a vertical wall.

The rectangular test cell (8.89 cm in height, 6.35 cm in width) was heated on one side and a heat sink was located on the opposite side wall. The other walls were insulated with Plexiglass, and the front and back sidewalls contained an air gap between two Plexiglass plates. These sidewalls were assumed to be adiabatic in the validation simulations. In the experiment used for this validation, the heat sink was held at a constant temperature of $28.3\,°C$, which is also the initial temperature of the gallium in the domain, whereas the heated wall was held at constant $38.0\,°C$. Since the mathematical model of this problem is naturally described with Dirichlet boundary conditions, prescribing a constant temperature at the walls, but Robin boundary conditions are implemented in the FEM model, the thermal conductivity at the wall was chosen to have a very high value $\alpha_{boundary} = 10^{10}\,\mathrm{W/mK}$.

The gallium used in the experiments [5] had a purity of 99.6 % and a melting temperature of $29.78\,°C$. All material properties needed for the simulation of this test case were extracted from the article [44], where a numerical investigation of exactly this experiment was conducted. A total of 26 equally spaced thermocouples were installed on the top and bottom walls and another 17 thermocouples along the center line, to measure the horizontal temperature distributions and therefore allow for tracing the interface of the melting front.

The shape of the melting front is given at nine different moments in the experiment, namely at minutes 2, 3, 6, 8, 10, 12.5, 15, 17 and 19 and was extracted from the graphic in the paper. From these melting fronts, the melted fraction in the gallium cavity was deduced and used as a quantity of comparison. Since the shape of the

melting front, calculated from simulation results, is not a mesh-independent solution, the liquid fraction is calculated via the current latent heat stored in the cavity. To draw comparisons, again the RMSE (root mean square error) value

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(a_{experiment} - a_{simulations}\right)^2} \tag{5.2}$$

is used with the $n = 9$ time comparison points, where a is the liquid fraction in the cavity.

5.2.2 Validation and convergence analysis

A convergence analysis was performed on the effect of mesh resolution, timestep size, mushy region temperature range and also aspect ratio on the simulation accuracy compared to the experimental solution in the explained test case (see Section 5.2.1). Thus, a validation for different values of these parameters is given.

Effect of mesh refinement

To study the effect of mesh refinement, the gallium test case, as described above, was simulated for different values of the mesh size, while the time step size $\Delta t = 25\,ms$ and mushy region parameter $\epsilon = 0.1\,°C$ were held constant. The mesh was chosen so that Δx=Δy. Therefore, the particular mesh size is labelled only by the value of Δx in the following. The RMSE compared to the experimental data and the RMSE compared to the simulation with $\Delta x = 0.5\,mm$ are given in Table 5.4.

With the RMSE compared to the most precise simulation with $\Delta x = 0.5\,mm$, we can observe a significant improvement from the coarsest mesh to the mesh with $\Delta x = 2.0\,mm$. From this value on, the convergence is very slow and the simulations do not considerably increase in accuracy compared to the experimental data.

Table 5.4: RMSE of liquid fraction for different values of mesh size or number of elements compared to test case (experimental) and $\Delta x = 0.5\,mm$ simulation (convergence)

Mesh size $\frac{\Delta x}{mm}$	4	2	1.6	1	0.8	0.5
Number of elements	352	1408	2200	5632	8800	22528
$\frac{\text{RMSE}}{100}$ (experimental)	4.72	2.32	2.26	2.31	2.16	1.95
$\frac{\text{RMSE}}{100}$ (convergence)	2.84	0.53	0.39	0.38	0.25	-

Effect of applied time step

The gallium test case was simulated for different time step sizes, while the mesh size $\Delta x = \Delta y = 1\,mm$ and mushy region temperature range $\epsilon = 0.1\,°C$ were held constant. The RMSE compared to the experimental data and the RMSE compared to the finest simulation with $\Delta t = 2\,ms$ are shown in Table 5.5.

Table 5.5: RMSE of liquid fraction for different values of the time step size compared to test case (experimental) and $\Delta t = 2\,ms$ simulation (convergence)

Time step size $\frac{\Delta t}{ms}$	50	25	15	10	5	2
$\frac{\text{RMSE}}{100}$ (experimental)	2.62	2.31	1.87	1.50	1.12	1.34
$\frac{\text{RMSE}}{100}$ (convergence)	2.77	2.63	2.22	1.65	0.94	-

Looking at the RMSE values, convergence of the simulation result with decreasing time step size was found, although not quite as fast as with refinement of the mesh. The same can be said about the improvement of accuracy compared to the experimental data. Still, a root mean square error of below 2 %, which accounts to a relative

deviation of around 3 % of the experimentally obtained liquid fraction, can be seen as a very good validation of the convection model.

Effect of mushy region size

The effect of the mushy region size ϵ on the simulation result of the gallium test case was examined under constant values of mesh size $\Delta x = 1\,mm$ and time step size $\Delta t = 25\,ms$. Again, the RMSE compared to the experimental data is shown in Table 5.6. The RMSE values in Table 5.6 are similar to the findings in Section 5.1. Again, the melting front lags slightly behind the compared solution, which in this case is the liquid fraction, for large values of the mushy region size, but it is not causing large errors. For small values of the mushy region size, the simulation gets poorer, for the same reasons already explained for the 1D validation. We can therefore conclude that when using the apparent heat capacity method for isothermal phase change problems, approximated by a very small mushy region, a very small time step size must also be used.

Table 5.6: RMSE of liquid fraction for different values of the mushy region size compared to test case (experimental)

Mushy region size $\frac{\epsilon}{^\circ C}$	0.5	0.25	0.1	0.05	0.02
$\frac{\text{RMSE}}{100}$ (experimental)	1.58	1.14	2.31	2.52	9.70

Effect of the aspect ratio

Another question with respect to stability and convergence of the developed model was whether large aspect ratios $\frac{\Delta x}{\Delta y}$ or $\frac{\Delta y}{\Delta x}$ cause any problems for the simulation. Such a choice of mesh might be convenient for simulation domains with large aspect ratios, because of the reduced total number of elements. To answer this question, the gallium test case was simulated for different values of the aspect ratio, while the time

step size $\Delta t = 25\,ms$ and mushy region parameter $\epsilon = 0.1\,°C$ were held constant. The melted volume fraction of such simulations, where the aspect ratio was tested with respect to larger values of Δx, is shown in Figure 5.2. The same analysis was done for larger values of Δy, and the melted volume fraction is shown in figure 5.3.

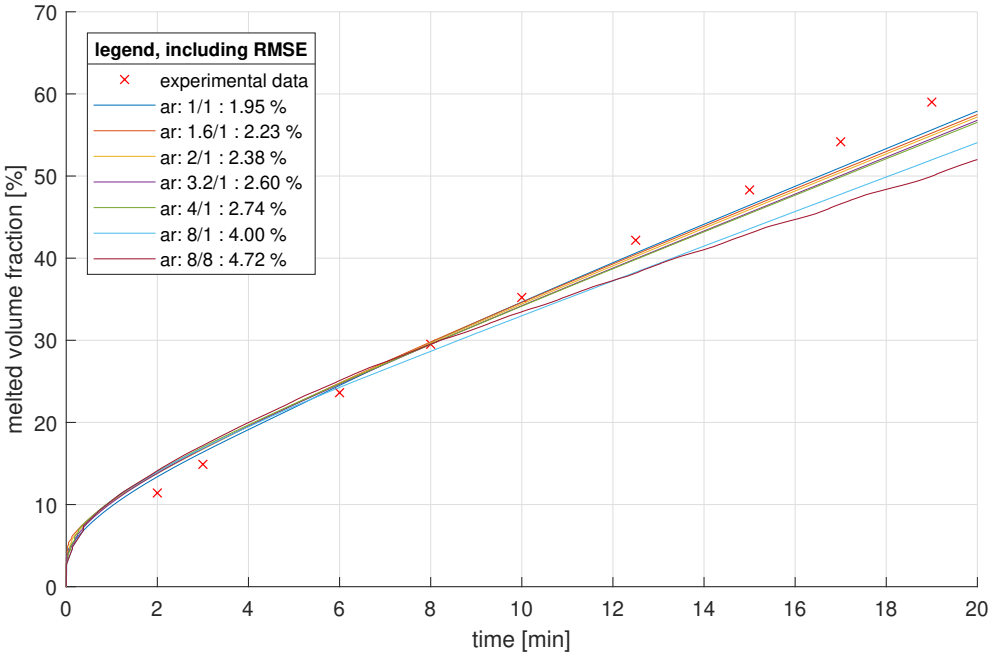


Figure 5.2: Time evolution of liquid fraction for different aspect ratio values $ar = \dfrac{\Delta x}{\Delta y}$ relative to the base case $\Delta x = \Delta y = 0.5\,mm$ compared to experimental data with variable Δx

It can be seen that the deviation from the experiment gets larger with increasing aspect ratio, but still does not exceed the range of error of the coarsest mesh $\Delta x = 4\,mm$, $\Delta y = 4\,mm$, which is denoted by the aspect ratio $8/8$ in both figures mentioned. Therefore, this approximation error can be traced back to the coarseness of the mesh, and no complications that come with large aspect ratios have been found.

Comparing Figures 5.2 and 5.3, we can obtain that the refinement of Δx has the same effect as the refinement of Δy. Furthermore, it can be observed that reasonably

fine meshes already produce good simulation results, which was previously stated in Section 5.2.2.

It should be noted though, that it cannot be ruled out that complications during simulations with large aspect ratios of the mesh could occur for different domain designs, and it is therefore recommended to test this in an analysis of mesh convergence, as was conducted in this chapter.

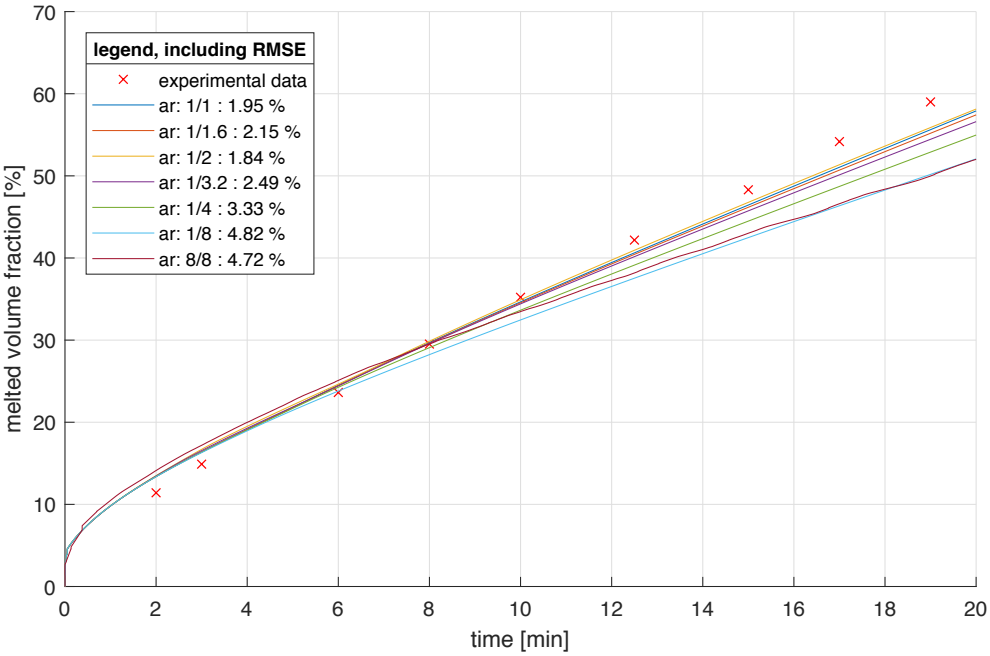


Figure 5.3: Time evolution of liquid fraction for different aspect ratio values $ar = \frac{\Delta x}{\Delta y}$ relative to the base case $\Delta x = \Delta y = 0.5\,mm$ compared to experimental data with variable Δy

Final validation results of the convection model

To conclude the validation of the convection model via the gallium test case, the shape of the melting front for selected simulations is discussed here.

In Figure 5.5, the melting fronts of two of the simulations already described in Section 5.2.2, with different mesh sizes Δx=Δy, a time step size of $\Delta t = 0.25\,ms$ and mushy

region parameter $\epsilon = 0.1\,°C$ are plotted at different comparison moments. Although the RMSE compared to the experimental data of these two simulations (see Table 5.4) was similar, there is still a significant difference in the shape of the developing melting fronts. The same is true for the two simulations with $\Delta t = 5\,ms$ and $\Delta t = 50\,ms$, both with $\Delta x{=}\Delta y = 1\,mm$ and $\epsilon = 0.1\,°C$, which are given in Figure 5.4. Interestingly, the shape of the melting front of the coarser simulation seems to fit the experimentally obtained melting front better than the one with the finer parameters. A possible explanation of this is the separation of physical and numerical effects, whereby expected modelling and discretization errors might cancel each other out on a coarse mesh compared to a finer mesh solution [45]. This effect of course is already well known in CFD simulations and was reported numerous times, in the reference [46], for example, where it was found that reasonably coarse meshes might yield better results compared to finer mesh structures and that further refinements might even deteriorate the results.

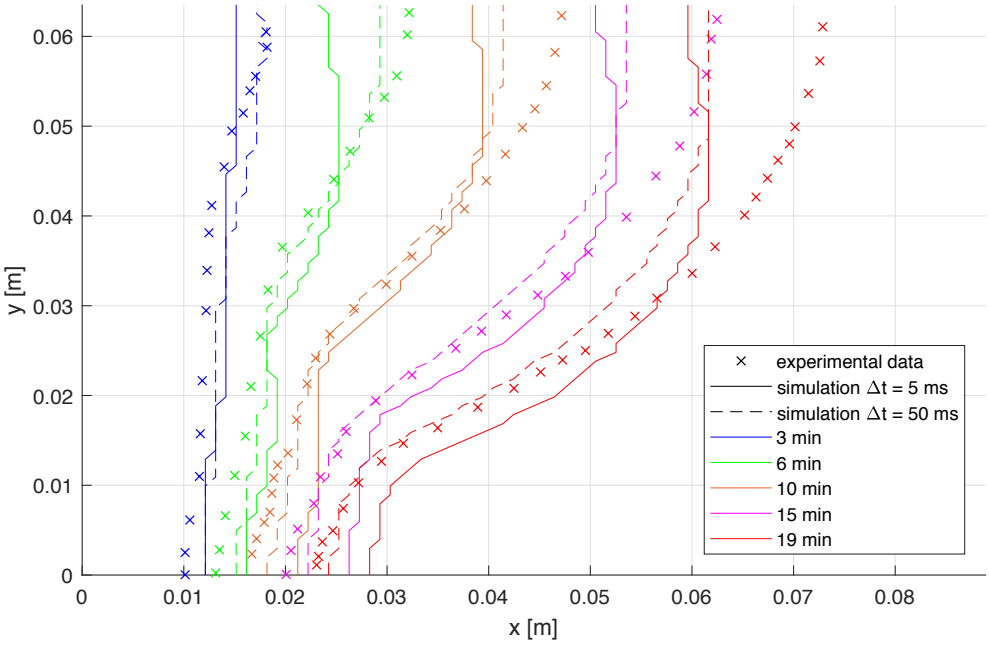


Figure 5.4: Melting fronts of two simulations with different time step size compared to experimental data, given at five time points

In spite of the peculiarities explained in this section, overall a good agreement of the model predictions with the experimental results published in [5] was found.

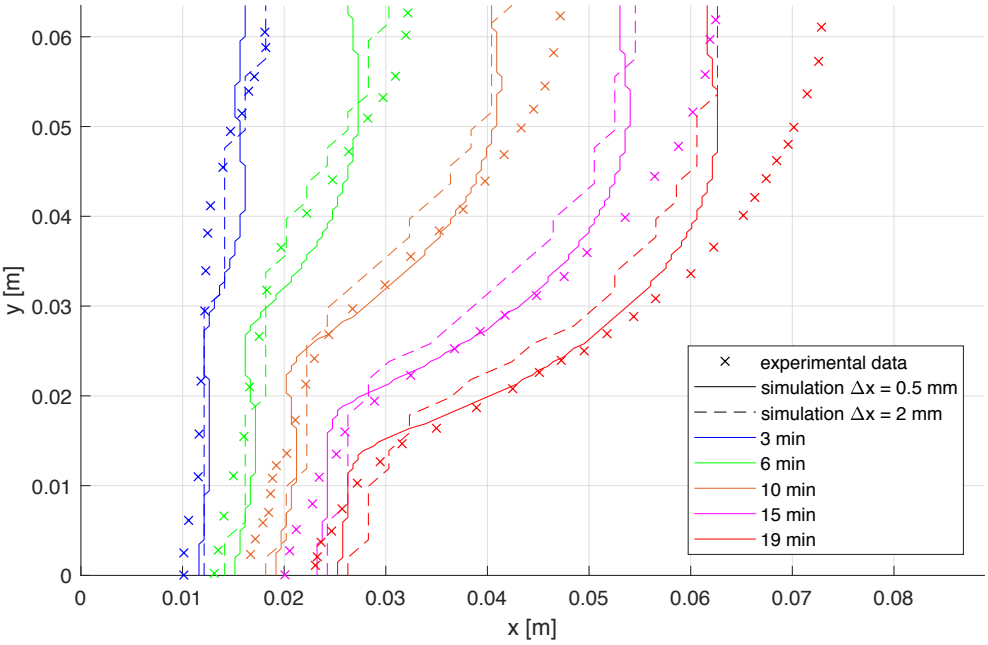


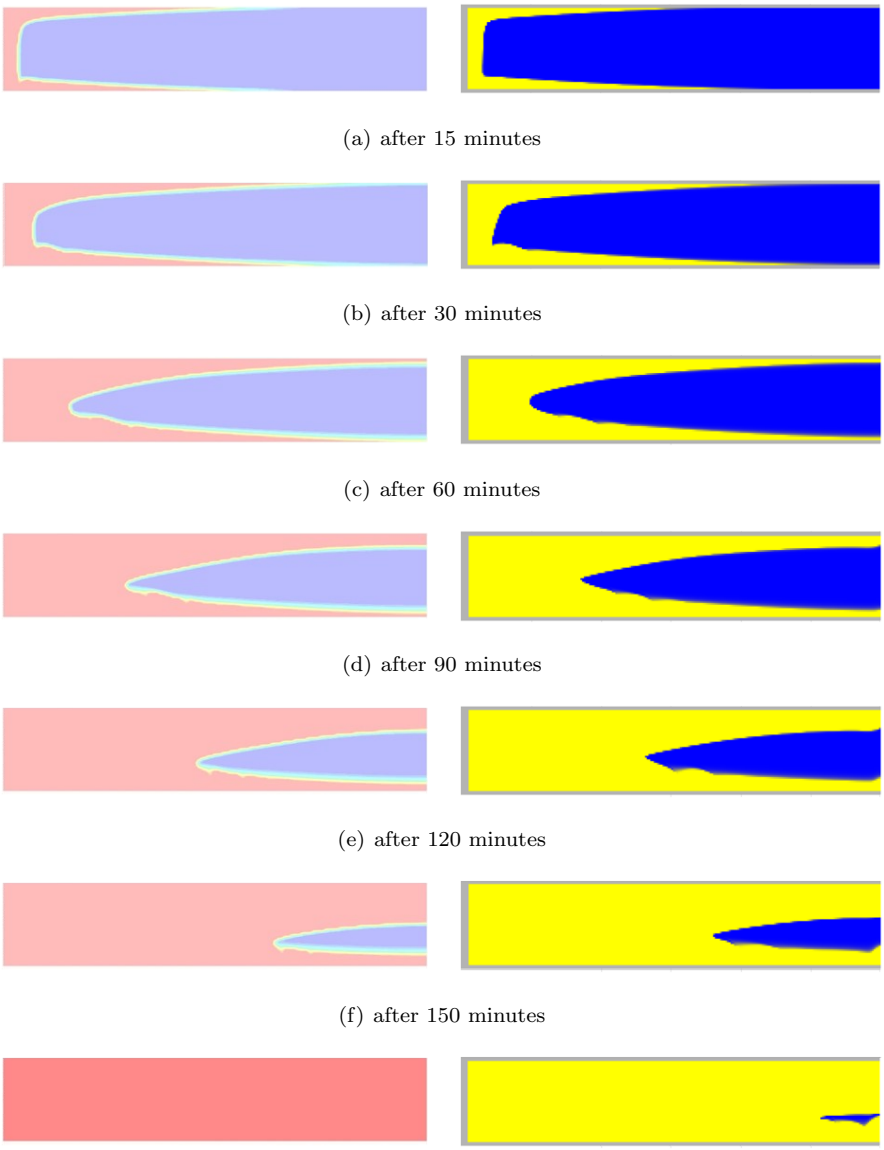
Figure 5.5: Melting fronts of two simulations with different mesh size compared to experimental data, given at five time points

5.3 Cross-validation with ANSYS Fluent

In the course of conducting the parameter study, which will be discussed in Chapter 6, simulations were carried out in the developed MATLAB model and also run in ANSYS Fluent. Thus, a cross-validation with the renowned CFD software is performed, a brief illustration of which is shown here.

In the cases which could be compared using the two simulation methods, a maximum relative deviation of 13 % at any time step in the simulated liquid fraction was found when compared to the prediction of Fluent. In general though, the observed discrepancy was considerably smaller than that. As an example, the comparison for the case of a standard PCM cavity (as will be described in Chapter 6) in horizontal position

and heated from the left side, shall be given here. Figure 5.6 shows the phase structure in the cavity at different times calculated by Fluent on the left side and calculated by the MATLAB model on the right side. Not only the extent of the liquid phase, but also the shape of the melting front, simulated by the developed FEM model, were found to be in very good agreement with the results from Fluent.



(a) after 15 minutes

(b) after 30 minutes

(c) after 60 minutes

(d) after 90 minutes

(e) after 120 minutes

(f) after 150 minutes

(g) after 180 minutes

Figure 5.6: Phase structure in PCM cavity at different times

Note. The solid region is portrayed in blue and the liquid region is portrayed in red for the Fluent simulation (left side) and yellow for the FEM simulation (right side).

5.4 Test of a fluctuating charging/discharging scenario

To determine the stability of the developed model in a simulation of operating latent heat hybrid storage, a test scenario with multiple irregular charging and discharging cycles was performed. The goal was to discern whether two or even more separated liquid cells could develop in this PCM storage and to observe whether any numerical problems thereby occur.

It should be pointed out that, the CFL condition, which is introduced when the nonlinear Navier-Stokes term is treated explicitly (see Section 3.3.2), is necessary for the stability of the simulation. This condition implies that the maximum velocity occurring in the domain must be smaller than the ratio of the spatial resolution to the time step size.

The geometry and material properties used in this examination are those of the gallium validation case, described in Section 5.2. The input temperature on the left side wall of the domain was specified differently, and in an irregularly fluctuating manner, which is shown as a dashed line in Figure 5.7. The mean temperature in the gallium cavity, as well as the maximum velocity appearing in the molten gallium, are presented in this Figure as solid blue and orange lines respectively.

It can be found that the velocity in the cavity builds up very quickly to the maximum value and is later diminished by declining driving forces, namely, temperature differences in the domain. This reaction can be seen for examples at around 7, 27 and 37 minutes, when the gallium is fully liquefied, and when the input temperature drops and a solid layer develops at the left side wall. Hence, no further adaption to the model to slow down the convective motion must be made. Furthermore, no numerical difficulties or 'unphysical behaviour' was observed during this test simulation scenario.

As to the development of separated liquid cells in the gallium cavity, no more than two of them could be observed at any time in the simulation. The dynamic of the

convective heat transfer leads to the separated zones merging very quickly and therefore makes more than two of these zones coexisting rather unlikely. Although these improbable scenarios might occur in very wide and flat cavities, the shape of the melting front studied later in Chapter 6 leads to the assumption that they will not occur in this kind of cavity with aluminium fins. An example snapshot of the temperature and velocity field during the merge of two separated liquid cells is shown in Figures 5.8 and 5.9. The solid fraction can be easily obtained from the representation of the velocity distribution in 5.9.

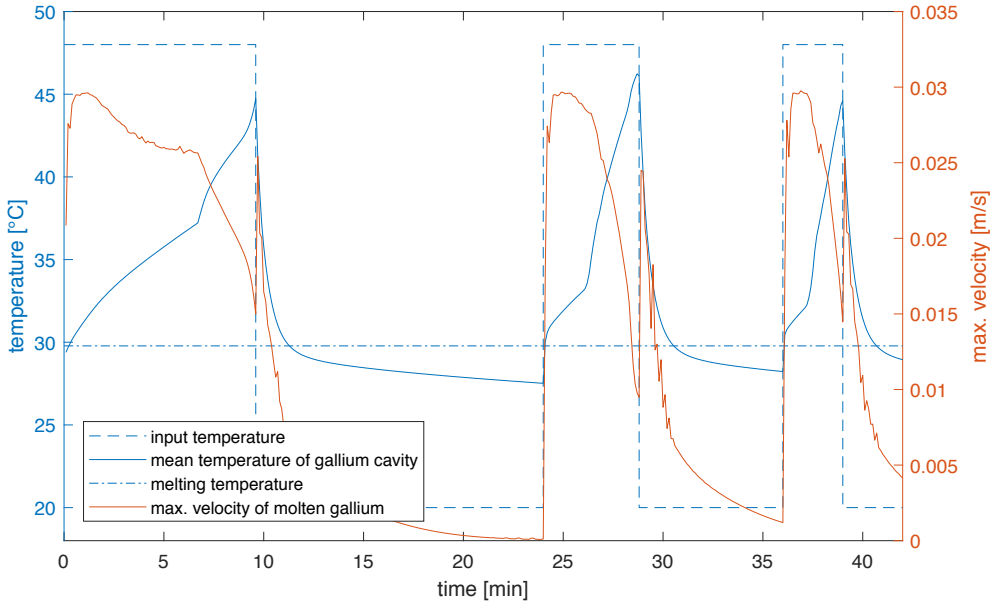


Figure 5.7: Temperature and velocity evolution over time in fluctuating gallium test case charging/discharging scenario

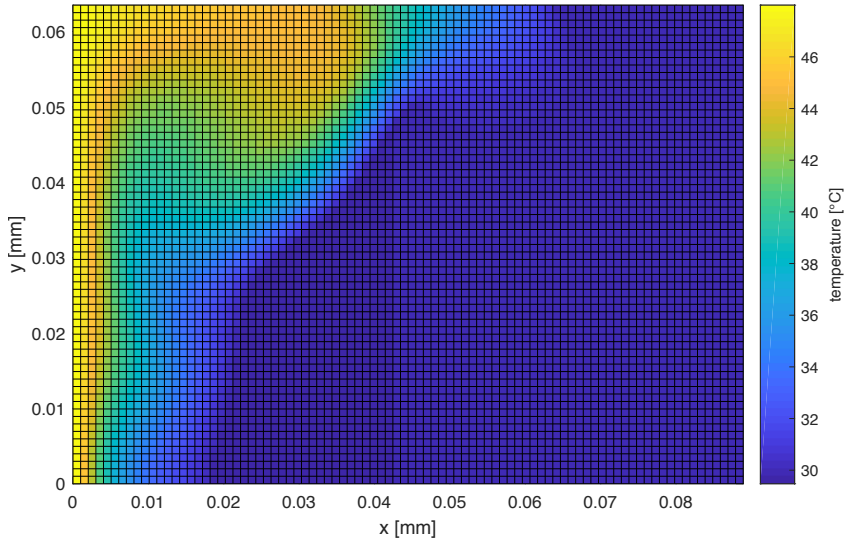


Figure 5.8: Temperature distribution in the gallium cavity at the simulation time of 26 minutes during charging/discharging scenario

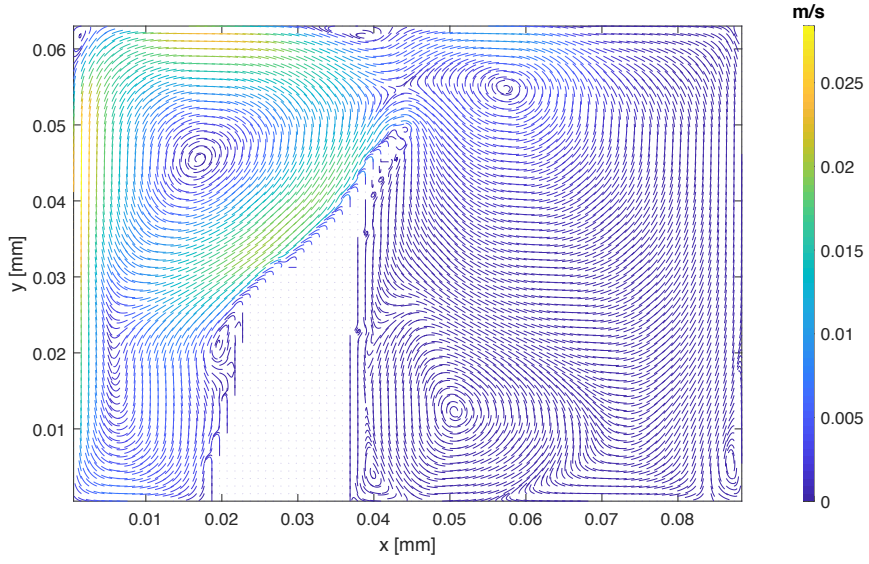


Figure 5.9: Velocity distribution in the gallium cavity at the simulation time of 26 minutes during charging/discharging scenario

Chapter 6

Parameter studies

As an example of what can be done with the developed FEM MATLAB model, a parameter study conducted to investigate different designs specifics of latent heat storage cavities is shown here. Also, the dynamics of melting and solidification for different orientations of a selected PCM cavity were examined for the charging and discharging mode. For this selected cavity design, the effect of natural convection is presented in contrast to simulations which do not consider convection.

6.1 Study of different cavity designs

As already mentioned in this thesis, a number of different designs are currently under consideration for the PCM cavity of the HyStEPs hybrid latent heat storage. At this point, the properties of the materials and operation parameters are not yet known. The parameter study of different cavity dimensions discussed in this section was conducted at a very early stage in the project. Many properties were still unknown or open for discussion. Also, natural convection was not yet implemented into the model, which is why this parameter study considers only heat conduction. Therefore, the simulations conducted and the results obtained might not be of great value for future considerations. However, some unique characteristics of the cavity design discussed here could also be found for different dimensions of the cavity simulated here. Furthermore, the methodical approach to accomplish first estimations of the behaviour of chosen designs can be adopted in the future.

The basic design of the latent heat storage in consideration is that of an aluminium-framed PCM cavity attached to the steel containment of the steam storage, while the outward-facing side of the cavity is insulated. Of course, the real life implementation will most probably look different due to engineering reasons, but nevertheless, should

be well approximated by this model. To enhance the heat transfer in the PCM cavity, aluminium fins should be placed in the plane of the axis of the cylindric steam storage. A sketch of such a fin section is shown in Figure 6.1.

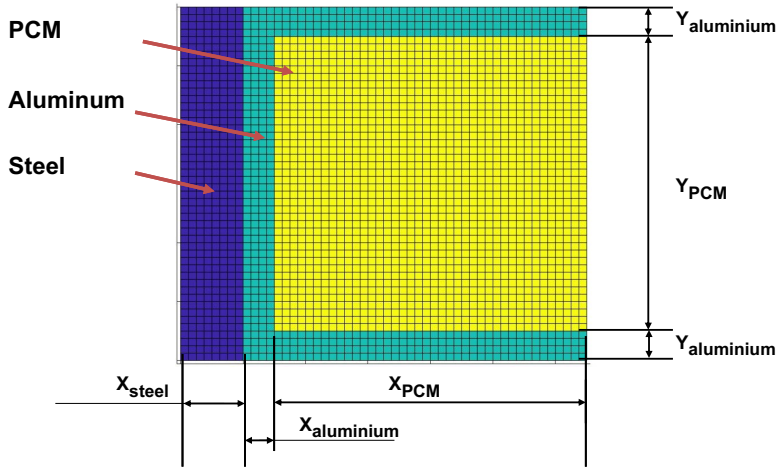


Figure 6.1: Sketch of a typical PCM cavity fin section attached to the steel containment of the steam storage

6.1.1 Simulated parameters

The simulation domain of the PCM cavity was defined as shown in Figure 6.1, with sections of different materials. The material properties are those shown in Table 3.1. The width of the steel containment was assumed to be $10\,mm$, and the heat transfer from the steam to the steel containment was described by the thermal conductivity $\alpha_{in} = 100\,\frac{W}{m^2K}$. The heat transfer through the well insulated outwards facing wall was described by the thermal conductivity $\alpha_{out} = 0.02\,\frac{W}{m^2K}$. Again, these parameters might be very different in reality and should be thoroughly discussed for future evaluations.

Regarding the computational parameters for these studies, a mesh of 80 times 80 elements was chosen. The mushy region parameter was set to $\epsilon = 2.0\,°C$, together with a time step size of $\Delta t = 10\,s$. The initial temperature in the domain was set to $190\,°C$ and the input temperature was set to $235\,°C$ for one hour of time and back to

$190\,°C$ for a second hour, which should represent a simplification of a typical operation cycle of the steam storage, where melting and solidification can be observed.

The aim of the conducted parameter study was to obtain an 'ideal' value of the geometric aspects of a fin section, meaning the dimensions of the PCM X_{PCM} and Y_{PCM} and the width of the aluminium fins $Y_{aluminium}$ for this kind of storage operation. The width of the aluminium between the steel containment and the PCM was set to a constant $X_{aluminium} = 5\,mm$ and the steel containment width was set to a constant $X_{steel} = 10\,mm$.

A total of 32 simulations were carried out, of which 16 simulated a fin width of $Y_{aluminium} = 5\,mm$ and 16 a fin width of $Y_{aluminium} = 2.5\,mm$. The dimensions of the PCM were varied between $X_{PCM}, Y_{PCM} = 5\,mm$ and $X_{PCM}, Y_{PCM} = 50\,mm$. The simulated cases together with the associated tagging are given in Tables 6.1 and 6.2.

Table 6.1: Simulated parameter study cases and associated tags with fin width $Y_{aluminium} = 5\,mm$

		$\frac{Y_{PCM}}{mm}$			
		50	25	10	5
$\frac{X_{PCM}}{mm}$	50	10v10	10v20	10v30	10v40
	25	20v10	20v20	20v30	20v40
	10	30v10	30v20	30v30	30v40
	5	40v10	40v20	40v30	40v40

Table 6.2: Simulated parameter study cases and associated tags with fin width $Y_{aluminium} = 2.5\,mm$

		$\frac{Y_{PCM}}{mm}$			
		50	25	10	5
$\frac{X_{PCM}}{mm}$	50	11v10	11v20	11v30	11v40
	25	21v10	21v20	21v30	21v40
	10	31v10	31v20	31v30	31v40
	5	41v10	41v20	41v30	41v40

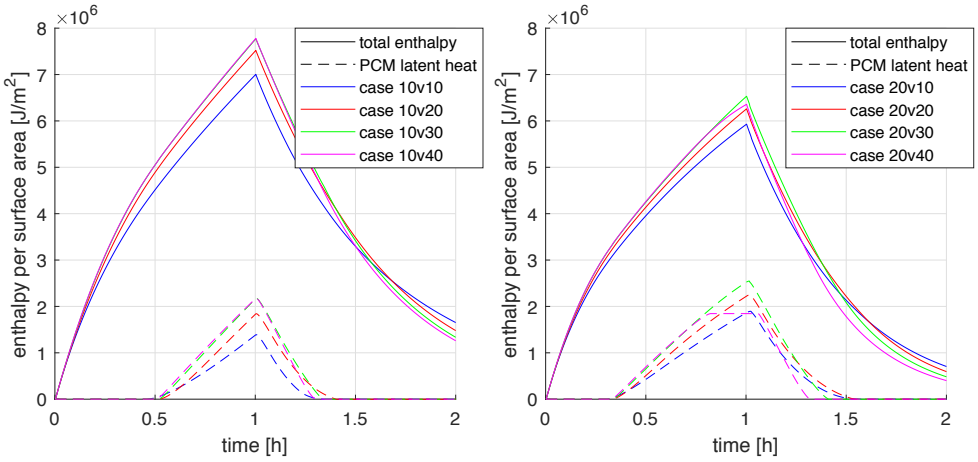
6.1.2 Simulation results

To analyse the simulation results and compare the different cavity dimensions, the evolution of the enthalpy change per surface area $\frac{\Delta H}{A}$ was calculated, where A is the surface area between the PCM fin section and the steam storage. The surface therefore takes the value

$$A = (Y_{PCM} + 2Y_{aluminium}) \cdot 1m \ . \tag{6.1}$$

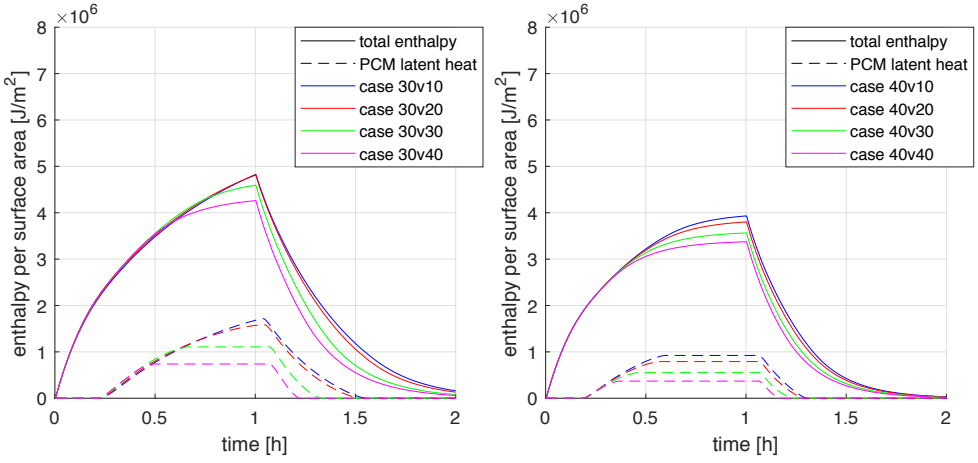
In Figures 6.2 and 6.3, this quantity and the latent heat fraction of it is plotted for the different simulated cases. In the parameter studies conducted here, the latent heat fraction of the total enthalpy change ΔH is evidently relatively small. This is because the containment of $10\,mm$ steel and $5\,mm$ aluminium was also considered, which offers a larger storage of sensible heat.

It can be obtained that, for the selection of longer cavities ($X_{PCM} = 50\,mm$, $25\,mm$), the lower cavities can store more enthalpy per surface area and the charging and discharging process is also quicker. This difference in speed of charging and discharging between the cavity heights can also be obtained for the shorter cavities, as long as



(a) cases with constant $X_{PCM} = 50\,mm$

(b) cases with constant $X_{PCM} = 25\,mm$



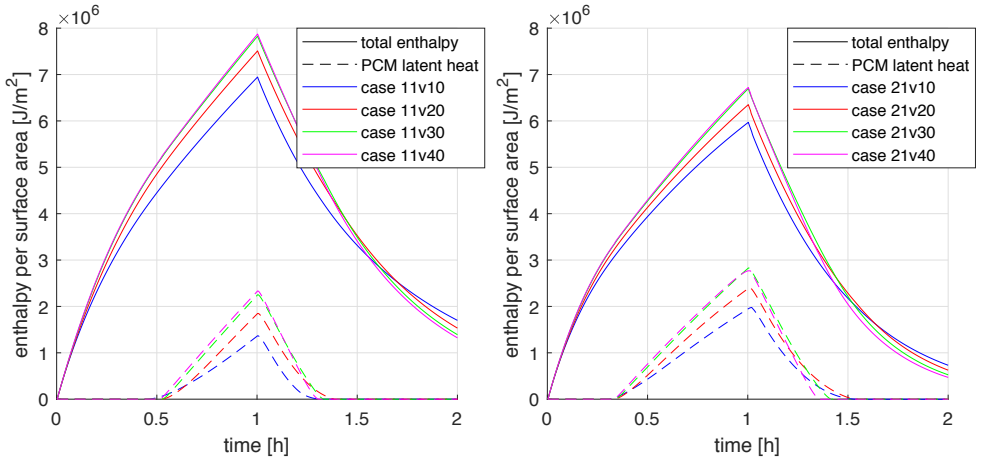
(c) cases with constant $X_{PCM} = 10\,mm$

(d) cases with constant $X_{PCM} = 5\,mm$

Figure 6.2: Enthalpy per surface area of the simulated cases with fin width $Y_{aluminium} = 5\,mm$

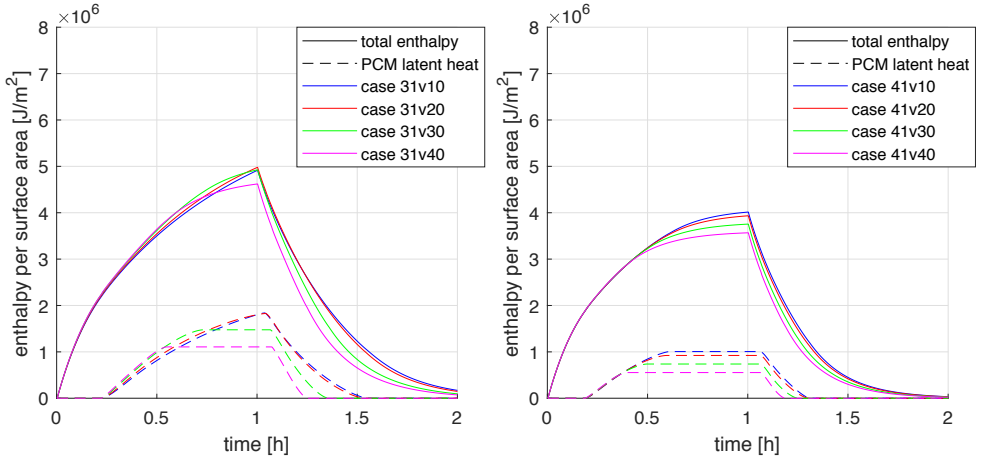
Note. The total enthalpy change ΔH is plotted using solid lines, the latent heat fraction using dashed lines.

the PCM is not fully liquefied. At that point, the higher cavities are able to store more enthalpy because of the higher ratio of PCM to aluminium. After all, the dimensioning of the cavity is also a question of cost. If unused PCM material must be



(a) cases with constant $X_{PCM} = 50\,mm$

(b) cases with constant $X_{PCM} = 25\,mm$



(c) cases with constant $X_{PCM} = 10\,mm$

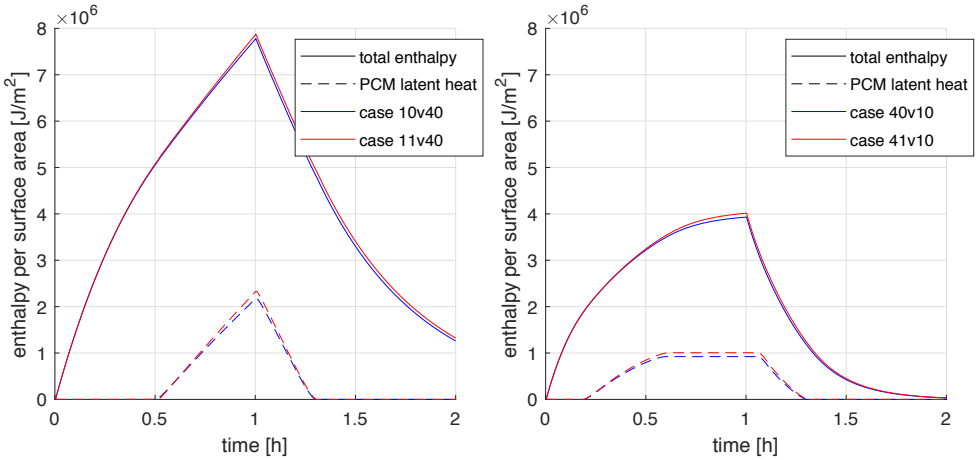
(d) cases with constant $X_{PCM} = 5\,mm$

Figure 6.3: Enthalpy per surface area of the simulated cases with fin width $Y_{aluminium} = 2.5\,mm$

Note. The total enthalpy change ΔH is plotted using solid lines, the latent heat fraction using dashed lines.

avoided because of budgetary reasons, a cavity as in case 41v10 should be preferred. If the material costs are not considered and the total storable enthalpy should be maximized, cavity dimensions as in case 11v40 are the optimal choice.

Looking at Figure 6.4, we can conclude that the smaller fin width of $Y_{aluminium} = 2.5\,mm$ should be preferred over the version with $Y_{aluminium} = 5.0\,mm$ because the over all storage capacity is higher and the charging/discharging speed is not diminished by the thin fins. This could of course be different for PCM materials with higher thermal conductivity, in which case the heat transfer could be limited by the width of the aluminium fins. However, convection, which leads to an effectively higher heat conductivity, could also provoke this effect. Finally, constructive constraints must also be considered, which generally include a lower limit for the minimum width of aluminium fins.



(a) cases with constant $X_{PCM} = 50\,mm$ and $Y_{PCM} = 5\,mm$

(b) cases with constant $X_{PCM} = 5\,mm$ and $Y_{PCM} = 50\,mm$

Figure 6.4: Enthalpy per surface area of the simulated cases with different fin width $Y_{aluminium}$

Note. The total enthalpy change ΔH is plotted using solid lines, the latent heat fraction using dashed lines.

Considering the width of the fins, it should be stressed that the results obtained in this study should not lead to the assumption that thinner fins are always the better choice. In fact, further investigations on the fin dimensions of a cavity, as described

in Section 6.2.2 with $X_{PCM} = 118\,mm$, showed a significant influence of the fin width on charging/discharging speed. Although these investigations are not particularized in this thesis, this should again emphasize the importance of further studies on the dimensioning of the HyStEPs cavity.

6.2 Comparison of different cavity orientations

In this section, the effect of the orientation of a typical PCM cavity with respect to its position on the steam storage is studied. The term 'typical' PCM cavity is of course very subjective, in that it is problem-dependent and based on many different assumptions. As mentioned earlier, the cavity dimensions studied in Section 6.1 were later found to be of little relevance for future application. Based on an assessment of fellow HyStEPs project partners, the amount of PCM needed to produce a desired latent heat storage capacity for a designated application case could be estimated and leads to a required PCM coating of the steam storage of around $120\,mm$ thickness. Thus, the geometry of an, at this point, more representative PCM cavity fin section, described in detail in the following, is simulated in this section.

6.2.1 Simulations in ANSYS Fluent

The simulations carried out here in the developed MATLAB model were also run in ANSYS Fluent. This was done to double-check the results and to have a second tool ready and parametrized for future simulations. In Section 5.3, the successful cross-validation was already reported. As the simulations in Fluent are not essentially part of this thesis, these will not be discussed in detail. It should only be noted, that all results obtained from the two simulation methods were very similar (see for example 5.6) and can be interpreted the same way.

Another observation worth mentioning is the negligible effect of small mushy regions for the kind of phase change problems simulated. A comparison of a typical parameter study case, as reviewed in this section and simulated in Fluent once with a mushy

region parameter of $0.1\,°C$ and once with isothermal phase change (which Fluent can handle), showed a relative difference in the results of less than $1.5\,\%$. This is a further convenience of the developed MATLAB model with regard to the apparent heat capacity method, because it means that isothermal materials can also be approximated fairly well by a very small mushy region.

6.2.2 Simulation parameters

The simulated domain still represents the PCM cavity fin section illustrated in Figure 6.1, except for the steel containment, which was not implemented in this study in order to lower the computational effort. This can be justified by the finding during the preliminary parameter study, explained in Section 6.1, that the steel containment has a negligible effect on the transient behaviour of the whole cavity. The dimensions of the fin section took fixed values in the simulations described here. The PCM was defined by $X_{PCM} = 118\,mm$ and $Y_{PCM} = 23\,mm$, the width of the aluminium fins was set to $Y_{aluminium} = 1\,mm$, and between the PCM and the steam storage an aluminium layer of $X_{aluminium} = 2\,mm$ was defined.

The material properties used for these simulations can again be found in Table 3.1. The heat transfer from the steam to the aluminium layer (via the neglected steel containment) was described by the thermal conductivity $\alpha_{in} = 700\,\frac{W}{m^2 K}$. The heat transfer through the outward facing wall was assumed as adiabatic.

Regarding the computational parameters, a convergence analysis with respect to mesh and time step size was performed. For the chosen mushy region parameter of $\epsilon = 0.1\,°C$, convergence was reached with a time step size of $\Delta t = 0.1\,s$. A mesh with quadratic elements of $0.5\,mm$ length was chosen for the simulations, which agreed best with the comparative results from ANSYS Fluent.

For the simulation of the charging process of the PCM fin section, the initial temperature in the domain was set to $218\,°C$ and the input temperature was set to $230\,°C$ for three hours of simulation time. For the discharging simulation, an input temperature of $200\,°C$ was assumed and the initial temperature set to $230\,°C$.

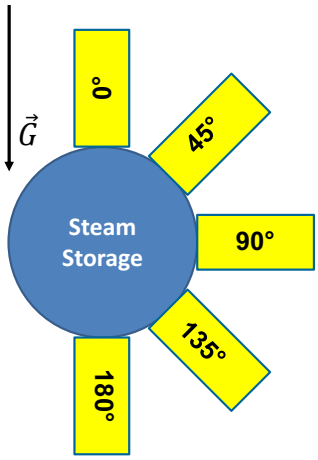


Figure 6.5: Schematic illustration of the five simulated cavity orientations in relation to the gravity vector

The simulations for charging and discharging of the PCM cavity, which included both the modelling of heat conduction and natural convection, were carried out for five different positions on the steam storage with characteristic angles in relation to the gravity vector. The five different orientations 0°, 45°, 90°, 135° and 180° are illustrated in Figure 6.5.

6.2.3 Simulation results

The change of total enthalpy during the charging process, as well as the latent heat fraction, are presented in Figure 6.6. Another representation of the simulation results is shown in Figure 6.7, where the evolution of the melted volume fraction during the charging and discharging process of the cavity is charted for the different orientations, as is the case in which only heat conduction is considered.

The horizontal cavity (90°) just reaches total liquefaction and, therefore, the fully charged state, only at the very end of the three hour simulation time, while for the upward facing cavities (0°, 45°), this state is already reached at two and a half hours. The downward facing cavities (135° and 180°) could not be completely charged

during the three hours of simulation time and a solid PCM fraction of more than 10 % remained. Furthermore, these two downward facing cavities show almost the exact same transient behaviour and only slightly outperform the charging process of the simulated case, where natural convection was neglected.

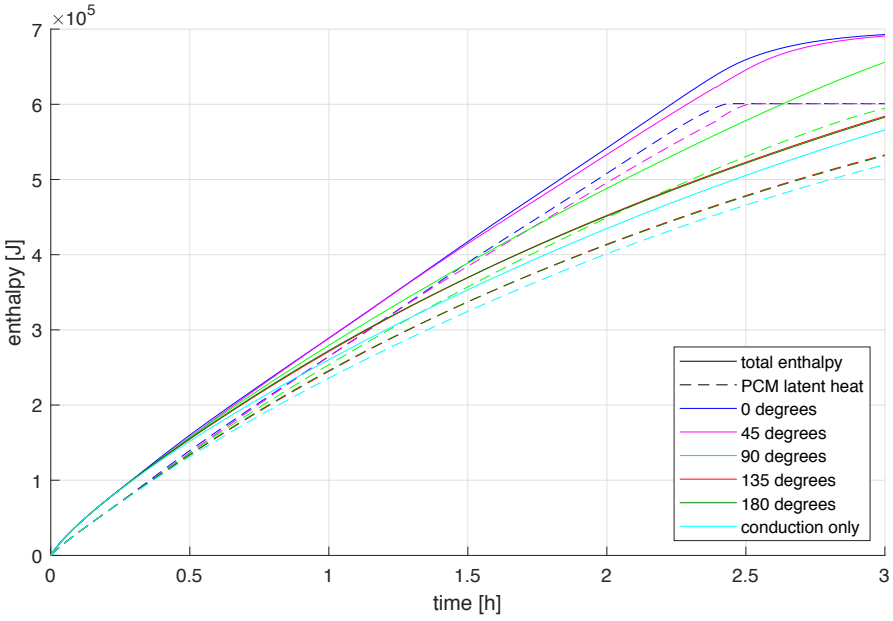
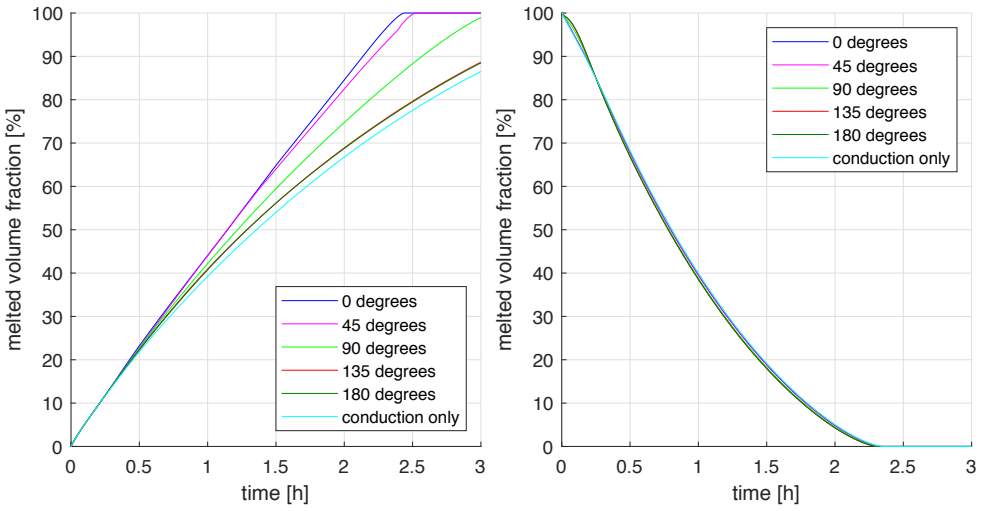


Figure 6.6: Change of enthalpy during the charging process

Note. The total enthalpy change ΔH is plotted using solid lines, the latent heat fraction using dashed lines.

Looking at Figure 6.7(b), it can be concluded that the orientation had no influence on the performance of the discharging process and that the latter is effectively the same as for the case in which natural convection was neglected.

In the appendix B, some plots of the temperature and velocity distribution of the simulation results for the five different cavity orientations are given, to illustrate the discussion in this chapter.



(a) charging

(b) discharging

Figure 6.7: Evolution of the melted volume fraction during charging (a) and discharging (b) of the cavity for different orientations

6.3 Discussion of conduction vs. convection simulations

Based on the results given in the previous section, we can conclude that the effect of natural convection on the charging process of the PCM cavity is highly dependent on the respective orientation. It cannot be neglected for upward facing cavities and is still a significant factor for cavities in horizontal position. For downward facing cavities however, the effect of natural convection is negligible and its behaviour can be described by heat conduction only, although it may be possible to improve accuracy by introducing a convective enhancement factor [47].

During discharging, the effect of natural convection was found to be insignificant, regardless of the cavity orientation.

Chapter 7

Conclusion

This last chapter presents a short summary of the overall approach to the stated problems and the main findings during the work on this diploma thesis. The findings are evaluated and put into perspective using previous research and possible applications. Finally, future investigation objectives and extensions to the developed MATLAB model are suggested.

7.1 Summary

The main task of this diploma thesis was the development of a MATLAB model, preferably via the finite element method, to solve phase change problems in typical latent heat storage geometries, considering both heat conduction and natural convection. To this end, comprehensive literature research was conducted to search for modelling techniques that are easily applicable to phase change problems, as occur in the HyStEPs hybrid latent heat storage. The necessary theoretical framework to understand the development of the numerical model was outlined in Chapter 2.

Chapter 3 explained the assumptions and discretization made to model the phase change problem. The simple-to-use apparent heat capacity method (see Section 2.2.2) was chosen to account for latent heat implementation in the material. To discretize the energy equation (3.1), the finite element method was applied, using a rectangular mesh of four-noded bilinear elements to approximate the simulated domain. For the modelling of convection via the Navier Stokes equation (3.4), an existing finite difference approximation method, documented in [36], was implemented, adapted to the problem and extended to fit the desired functionality. It should be noted though, that this code was used after unsuccessful attempts to discretize the Navier-Stokes equation by finite elements, using a penalty formulation for the pressure. In this case,

the effort to get a convergent numerical solution unfortunately failed. Nonetheless, this should not discourage fellow programmers from trying to apply this method for similar problems, as it was already done in [11].

A solid MATLAB implementation of the model explained in Chapter 3 was developed and documented in Chapter 4. This code can handle any desired layout of materials arranged on a rectangular domain, which is especially useful for PCM storage modelling with internal fins. The programming also aimed to achieve a high degree of vectorization to minimize the computational effort. While simulations considering only heat conduction considered lead to very fast solutions, the sparsity patterns of the matrix equation systems to solve cannot be exploited the same way for simulations considering natural convection. For large numbers of elements, the large systems of equations to be solved decreases the computational performance significantly.

The simulation results could be successfully validated in three steps, as documented in Chapter 5. The modelling of heat conduction could be verified as very accurate, when compared to the analytical solution of the one-dimensional Stefan problem. Depending on the choice of computational parameters, relative errors of the evaluated quantities of well under 1 % could be obtained. Also, fast convergence of the solution with decreasing time step size and refinement of the mesh size were observed.

Regarding the validation of the convection model, a good agreement of the model predictions with the experimental results published in [5] was found, whilst the results of the mesh convergence analysis were not as promising as those for the one-dimensional heat conduction model.

Additionally, a cross-validation of the convection model, given by simulations run in the CFD software ANSYS Fluent, was presented in Section 5.3, which also showed only small discrepancies between the two simulation methods.

Finally, parameter studies conducted with the developed MATLAB model, are given in Chapter 6. Different cavity designs were analysed with respect to total enthalpy input per surface area on the steam storage, while sustaining a fast charging and discharging speed.

Different cavity orientations were studied with respect to their position on the steam storage, which, in the charging process, showed a significant influence of natural convection for upwards facing cavities and horizontal cavities, but diminishing influence for downwards facing cavities. During discharging, the effect of natural convection was found to be insignificant, regardless of the cavity orientation.

7.2 Scope and limitations

The developed MATLAB model has proven to be an effective tool for modelling phase change problems, considering both conduction and natural convection heat transfer, in typical PCM cavities. This was shown by conducting informative parameter studies, which provide an example of the methodology for further investigations.

The code can also be used as a basis for model order reduction, which will be carried out by fellow research colleagues. This will ultimately lead to a reduced order model, which is essential for the real-time operation and control of the HyStEPs hybrid latent heat storage.

In comparison to commercially available code, such as ANSYS Fluent, the developed MATLAB model provides high usability in regards to running and evaluating parameter studies, which can be run parallel on any desired number of sessions/computing units. Furthermore, modifications to the existing code can be easily made, aiding model order reduction by using MATLAB directly. This is a major benefit for future work carried out in the HyStEPs project.

Having said this, some issues remain with the current MATLAB model. The nature of the apparent heat capacity method brings some difficulties in handling isothermal phase change problems. Although it could be obtained from Fluent simulations (see Section 6.2.1) that isothermal phase change can be approximated well by a very small mushy region, these simulations still require careful selection of computational parameters. As advised in the validation in Section 5.1, the magnitudes of the time step and mushy region size should ensure that the area of the mushy region progresses by overlapping its previous position. Otherwise, finite elements would skip the phase

change and would not be exposed to the effect of the latent heat. Even if these limitations are considered, an approximation error still remains, which is not the case for certain implementations of the enthalpy method, introduced in Section 2.2.1. Another option is source-based methods, in which the latent heat evolution is accounted for by the definition of a source term, instead of accounting for the latent heat in the specific heat coefficient. Such an approach was followed for some time to further improve the model capabilities, though it could not completely be put into practice, for reasons of limited time during this work. However, it is highly suggested to take up this approach again, which is outlined in more detail in the next section (see Section 7.3.3).

Although the parameter studies reported in Chapter 6 were based on preliminary design and material choices of the HyStEPs cavity, at least the methodological approach can be used for the purpose of future parameter studies. The findings of Section 6.3, with regard to the effect of natural convection in a typical PCM cavity, could prove to be a valuable insight for the development of the operating and control strategy in the HyStEPs project. For example, these results indicate a far less complicated charging behaviour for some cavity orientations, which implies a reduced number of sensors to be placed in the latter.

The observed effects can of course occur very differently for another PCM material or a modified cavity design. PCM materials with higher viscosity might lead to larger differences between the simulated cavity orientations, as will be opposite for low viscosity substances. The aspect ratio of PCM cavities has a significant effect on the heat transfer performances, as shown in reference [48], where, based on computational results, it was obtained that the melting rate increases as the aspect ratio of a rectangular cavity increases.

The reported effect of natural convection during charging and the sufficient approximation of the discharging process by only considering heat conduction was also reported in reference [27] for similar PCM cavity geometry.

7.3 Suggested future objectives

7.3.1 Subsequent parameter studies

As mentioned before, the design of the latent heat cavities, which will be mounted on a Ruths steam storage as part of the HyStEPs project, is not fixed at this time. Therefore, the parameter studies conducted in this thesis may not be particularly valuable for further investigations. Further parameter studies like the ones conducted in this thesis are suggested for the final design decision and to thoroughly analyse the characteristics of the chosen PCM cavity.

Building on the results of work, a more thorough parameter study is currently underway to provide a detailed foundation for design optimization of the considered PCM cell geometry [49]. Therein, varying aluminium proportions, varying fin spacings and different orientations of the PCM cell were simulated and their impact on charging/discharging speed was analysed.

7.3.2 Improvements of the existing MATLAB model

Some improvements to the developed MATLAB model may help reduce computational effort while maintaining the same level of accuracy. One suggestion is to implement adaptive time stepping, like for example presented in [50], instead of a fixed time step. Choosing a lager time step automatically could help speed up simulations in which temperature gradients are small and fluid flow is slow.

Another option is to specify separate fixed time steps or adaptive time step schemes for the finite element heat conduction problem and the finite difference Navier-Stokes solution method. Since the time step is generally limited by the CFL condition and not by the desired accuracy of the much faster FEM solution method, this would probably only lead to slightly better computational performance.

Furthermore, adaptive mesh or even moving mesh discretization methods, as documented in [11], could be considered for implementation. However, in this case, many of the faster preprocessing methods in the current code could note be used, and the benefit to performance is questionable

7.3.3 Fundamental modifications to the existing MATLAB model

Source-based method

As mentioned in Section 7.2, the implementation of a source-based method was attempted for some time during this work. Although the existing code was altered by the guidelines given in several references ([51] and [52]), the outcome was not satisfying since the simulation results were not reliable. For further reference, a brief explanation of the source-based method, is given here. This method could be especially important for accurately modelling isothermal phase change problems.

In the source-based method, the governing energy equation (2.5) for the apparent heat capacity method is written as

$$\frac{\partial(\rho c T)}{\partial t} = \nabla(k \nabla T) - \nabla(\rho c T \cdot \mathbf{u}) + S \ . \tag{7.1}$$

In this form however, c takes a fixed value instead of the apparent heat capacity, and the source term

$$S = -\delta H \frac{\delta a}{\delta t} \tag{7.2}$$

is added to the equation. Applying Galerkin's method to this equation, as done in Section 3.2.2, yields again a finite element system of the form

$$[C] \left\{ \dot{T} \right\} + [K] \{T\} = [R] \ . \tag{3.23}$$

Here, the source term is incorporated in the system matrices $[K]$ and $[R]$, and the equation system (3.23) can be solved in each time step by an iterative scheme as highlighted in the following:

1. The source term (7.2) is approximated by using an estimate of the liquid fraction field a.

2. Equation (3.23) can be evaluated and solved by a backwards Euler step.

The obtained solution for the temperature field will not necessarily be consistent with the current liquid fraction field. For instance, a predicted nodal temperature $T \neq T_m$ would not be consistent with a liquid fraction $a \in (0, 1)$. That is why the value of the nodal liquid fraction field is updated so that on subsequent iterations of step 1 and 2 the predicted nodal temperature is 'driven' to T_m, until a desired accuracy is reached.

Besides the approximation of the source term, the key to the source-based iteration is the method of updating the liquid fraction a. Numerous schemes for this purpose are found in literature, of which the reference [51] is emphasized here.

Enthalpy method

A different approach to the phase change problem would be to consider enthalpy H as dependent variable, instead of the temperature T. Although this approach would require serious modifications to the existing MATLAB implementation, it is possible to build on this work and reuse the existing formalism. For further reference, a good example of a finite element implementation of the enthalpy method can be found in [9].

Bibliography

[1] R. Hofmann, S. Dusek, S. Gruber and G. Drexler-Schmid. Design Optimization of a Hybrid Steam-PCM Thermal Energy Storage for Industrial Applications. *Energies*, 12(5), 2019. doi:10.3390/en12050898.

[2] S. Gruber. Charakterisierung eines Hybridspeichers zur Dampfbereitstellung in industriellen Prozessen. Theoretische Untersuchung und Konzeption eines Pruefstandes. Master's thesis, TU Wien, Vienna, Austria, 2018. URL http://katalog.ub.tuwien.ac.at/AC15194551.

[3] Y. Dutil, D.R. Rousse, N. Salah, S. Lassue and L. Zalewski. A review on phase-change materials: Mathematical modeling and simulations. *Renewable and Sustainable Energy Reviews*, 15:112–130, 2011. doi:10.1016/j.rser.2010.06.011.

[4] MATLAB. *version 9.3.0 (R2017b)*. The MathWorks Inc., Natick, Massachusetts, 2017.

[5] C. Gau and R. Viskanta. Melting and Solidification of a Pure Metal on a Vertical Wall. *Journal of Heat Transfer*, 108, 02 1986. doi:10.1115/1.3246884.

[6] A. Kumar and S.K. Shukla. A Review on Thermal Energy Storage Unit for Solar Thermal Power Plant Application. *Energy Procedia*, 74:462 – 469, 2015. ISSN 1876-6102. doi:10.1016/j.egypro.2015.07.728.

[7] K.K. Pillai and B.J. Brinkworth. The storage of low grade thermal energy using phase change materials. *Applied Energy*, 2(3):205 – 216, 1976. ISSN 0306-2619. doi:10.1016/0306-2619(76)90025-8.

[8] P. W. Egolf and H. Manz. Theory and modeling of phase change materials with and without mushy regions. *International Journal of Heat and Mass Transfer*, 37(18):2917 – 2924, 1994. ISSN 0017-9310. doi:10.1016/0017-9310(94)90346-8.

[9] B. Nedjar. An enthalpy-based finite element method for nonlinear heat problems involving phase change. *Computers and Structures*, 80(1):9 – 21, 2002. ISSN 0045-7949. doi:10.1016/S0045-7949(01)00165-1.

[10] W. Demtröder. *Experimentalphysik 1: Mechanik und Wärme*. Springer-Lehrbuch. Springer Berlin Heidelberg, 2018. doi:10.1007/978-3-662-54847-9.

[11] R.T. Tenchev, J.A. Mackenzie, T.J. Scanlon and M.T. Stickland. Finite element moving mesh analysis of phase change problems with natural convection. *International Journal of Heat and Fluid Flow*, 26(4):597 – 612, 2005. doi:10.1016/j.ijheatfluidflow.2005.03.003.

[12] S. Chandrasekhar. Stochastic problems in physics and astronomy. *Reviews of modern physics*, 15:1–89, 1943. doi:10.1103/RevModPhys.15.1.

[13] S. Braun. *Strömungslehre für TPh*. TU Wien, Institut für Strömungsmechanik und Wärmeübertragung, 2017.

[14] P. Dawkins. The Heat Equation. Available at http://tutorial.math.lamar.edu/Classes/DE/TheHeatEquation.aspx.

[15] T. Jonsson. On the one dimensional stefan problem : with some numerical analysis, 2013. URL http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A647481&dswid=-7335.

[16] T.F. Cheng. Numerical analysis of nonlinear multiphase Stefan problems. *Computers & Structures*, 75(2):225 – 233, 2000. ISSN 0045-7949. doi:10.1016/S0045-7949(99)00071-1.

[17] S. Liu, Y. Li and Y. Zhang. Mathematical solutions and numerical models employed for the investigations of PCMs phase transformations. *Renewable and Sustainable Energy Reviews*, 33:659 – 674, 2014. ISSN 1364-0321. doi:10.1016/j.rser.2014.02.032.

[18] H.J. Stetter. *Analysis of Discretization Methods for Ordinary Differential Equations*. Springer Tracts in Natural Philosophy. Springer Berlin Heidelberg, 2013. doi:10.1002/zamm.19750550213.

[19] D.W. Pepper and J.C. Heinrich. *The Finite Element Method: Basic Concepts and Applications*. Series in Computational and Physical Processes in Mechanics and Thermal Sciences. CRC Press, 2005. doi:10.1201/9780203942352.

[20] M. Kaltenbacher. *Numerical Simulation of Mechatronic Sensors and Actuators: Finite Elements for Computational Multiphysics, third edition*. Springer Berlin Heidelberg, 01 2015. doi:10.1007/978-3-642-40170-1.

[21] Mathematik Uni Dortmund. Finite difference method. Available at http://www.mathematik.uni-dortmund.de/~kuzmin/cfdintro/lecture4.pdfx.

[22] P.D.M. Causon, P.C.G. Mingham and L. Qian. *Introductory Finite Volume Methods for PDEs*. Bookboon, 2011. ISBN 978-87-7681-882-1.

[23] F. Moukalled, L. Mangani and M. Darwish. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab*. Fluid Mechanics and Its Applications. Springer International Publishing, 2015. doi:10.1007/978-3-319-16874-6.

[24] R.D. Richtmyer and K.W. Morton. *Difference methods for initial-value problems*. Interscience tracts in pure and applied Mathematics. Interscience Publishers, 1967. doi:10.1137/1010073.

[25] S. Dusek and R. Hofmann. A Hybrid Energy Storage Concept for Future Application in Industrial Processes. *Thermal Science*, 22(5):2235 – 2242, 2018. doi:10.2298/TSCI171230270D.

[26] M. M. Farid, A. M. Khudhair, S. A. K. Razack and S. Al-Hallaj. A review on phase change energy storage: materials and applications. *Energy Conversion and Management*, 45(9):1597 – 1615, 2004. ISSN 0196-8904. doi:10.1016/j.enconman.2003.09.015.

[27] J. Vogel, M. Johnson, M. Eck and D. Laing. Numerical analysis of natural convection in a latent heat thermal energy storage system containing rectangular enclosures. 2014. URL https://elib.dlr.de/113475/.

[28] J. Lopez, Z. Acem and E. P. Del Barrio. KNO3/NaNO3 - Graphite materials for thermal energy storage at high temperature: Part II. - Phase transition properties. *Applied Thermal Engineering*, 30(13):1586 – 1593, 2010. ISSN 1359-4311. doi:10.1016/j.applthermaleng.2010.03.014.

[29] D.R. Lide. *CRC Handbook of Chemistry and Physics, 85th Edition*. Number Bd. 85 in CRC Handbook of Chemistry and Physics, 85th Ed. Taylor & Francis, 2004. ISBN 9780849304859.

[30] S. Dusek and R. Hofmann. Modeling of a Hybrid Steam Storage and Validation with an Industrial Ruths Steam Storage Line. *Energies*, 12(6), 2019. ISSN 1996-1073. doi:10.3390/en12061014.

[31] D. Pernsteiner, L. Kasper, A. Schirrer, R. Hofmann and S. Jakubek. Co-simulation methodology of a hybrid latent-heat thermal energy storage unit. *Applied Thermal Engineering*, page 115495, 2020. ISSN 1359-4311. doi:10.1016/j.applthermaleng.2020.115495.

[32] H. Herr. *Wärmelehre: Technische Physik Band 3*. Europa-Lehrmittel, 4 edition, 2006. ISBN 978-3-8085-5064-9.

[33] M.J. Huang, P.C. Eames and B. Norton. Comparison of a small-scale 3D PCM thermal control model with a validated 2D PCM thermal control model. *Solar Energy Materials and Solar Cells*, 90(13):1961 – 1972, 2006. ISSN 0927-0248. doi:10.1016/j.solmat.2006.02.001.

[34] C. H. Stolze. A history of the divergence theorem. *Historia Mathematica*, 5(4): 437 – 442, 1978. ISSN 0315-0860. doi:10.1016/0315-0860(78)90212-4.

[35] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*. Applied mathematics series. Dover Publications, 1965. ISBN 9780486612720.

[36] B. Seibold. A compact and fast Matlab code solving the incompressible Navier-Stokes equations on rectangular domains, 2018. Available at http://math.mit.edu/~gs/cse/codes/mit18086_navierstokes.pdf.

[37] H. P. Langtangen, K. A. Mardal and R. Winther. Numerical methods for incompressible viscous flow. *Advances in Water Resources*, 25:1125–1146, 08 2002. doi:10.1016/S0309-1708(02)00052-0.

[38] R. Courant, K. Friedrichs and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100(1):32–74, 1928. doi:10.1007/BF01448839.

[39] Massachusetts Institute of Technology. Stability of Finite Difference Methods. Available at http://web.mit.edu/16.90/BackUp/www/pdfs/Chapter14.pdf.

[40] M. Griebel, T. Dornsheifer and T. Neunhoeffer. *Numerical Simulation in Fluid Dynamics: A Practical Introduction*. Monographs on Mathematical Modeling and Computation. Society for Industrial and Applied Mathematics, 1998. doi:10.1137/1.9780898719703.

[41] J. Dallaire and L. Gosselin. Numerical modeling of solid-liquid phase change in a closed 2D cavity with density change, elastic wall and natural convection. *International Journal of Heat and Mass Transfer*, 114:903 – 914, 2017. ISSN 0017-9310. doi:10.1016/j.ijheatmasstransfer.2017.06.104.

[42] GP Nikishkov. Introduction to the Finite Element Method. *University of Aizu*, 2009. Available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.521.9293&rep=rep1&type=pdf.

[43] K. Atkinson and W Han. *Elementary numerical analysis*. John Wiley & Sons, 1993. ISBN 9780471897330.

[44] A. D. Brent, V. R. Voller and K. J. Reid. Enthalpy-porosity technique for modeling convection-diffusion phase change: Application to the melting of a pure metal. *Numerical Heat Transfer*, 13(3):297–318, 1988. doi:10.1080/10407788808913615.

[45] M. Breuer. A challenging test case for large eddy simulation: high Reynolds number circular cylinder flow. *International Journal of Heat and Fluid Flow*, 21 (5):648 – 654, 2000. ISSN 0142-727X. doi:10.1016/S0142-727X(00)00056-4.

[46] Z. Boz, F. Erdogdu and M. Tutar. Effects of mesh refinement, time step size and numerical scheme on the computational modeling of temperature evolution during natural-convection heating. *Journal of Food Engineering*, 123:8 – 16, 2014. ISSN 0260-8774. doi:10.1016/j.jfoodeng.2013.09.008.

[47] J. Vogel, J. Felbinger and M. Johnson. Natural convection in high temperature flat plate latent heat thermal energy storage systems. *Applied Energy*, 184:184–196, 2016. doi:10.1016/j.apenergy.2016.10.001.

[48] W.B. Ye. Thermal and hydraulic performance of a rectangular storage cavity. *Applied Thermal Engineering*, 93:1114 – 1123, 2016. doi:10.1016/j.applthermaleng.2015.10.083.

[49] L. Kasper, D. Pernsteiner, M. Koller, A. Schirrer, S. Jakubek and R. Hofmann. Numerical studies on the melting and solidification process in an rectangular aluminium finned latent-heat thermal energy storage. in submission. *Applied Thermal Engineering*, 2020. Manuscript submitted for publication, under review.

[50] D. Kavetski, P. Binning and S. W. Sloan. Adaptive backward Euler time stepping with truncation error control for numerical modelling of unsaturated fluid flow. *International Journal for Numerical Methods in Engineering*, 53(6):1301–1322, 2002. doi:10.1002/nme.329.

[51] V.R. Voller and C.R. Swaminathan. Eral Source-Based Method for Solidification Phase Change. *Numerical Heat Transfer Part B - Fundamentals*, 19, 04 1991. doi:10.1080/10407799108944962.

[52] V.R. Voller, C.R. Swaminathan and B. Thomas. Fixed Grid Techniques for Phase Change Problems: A Review. *International Journal for Numerical Methods in Engineering*, 30:875 – 898, 09 1990. doi:10.1002/nme.1620300419.

Appendix A

MATLAB code listings

This appendix contains listings of the main functions of the developed MATLAB model.

Code Listing A.1: Code of main function PCMsolver

```
%% PCM Solver
% Author:            Lukas Kasper
% Last modified:     03.05.2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function PCMsolver(input_data)
dbstop if error;
%clearvars -except input_data;
%close all;

strTimestamp = datestr(now(),30); tic; % get timestamp
fprintf('Starting simulation / timestamp %s\n',strTimestamp);

%% preprocessing
% load input variables
eval(input_data);
% additional input variables
non = param.mesh.non;
not = param.mesh.not;

% initialize waittime bar
if param.display.waitbar == 1
        f = waitbar(0,'Fortschritt'); timearray = 0; up = 0;
end

% mesh generation
[param] = mesh2d(param);
v2struct(param.geom);
v2struct(param.mesh);
v2struct(param.icbc);

% visualize material in domain
if param.display.matplot == 1
    scatterbar3(center_coord(1,:)',center_coord(2,:)',...
                elem_mat,x_length/max(nx,ny));
end

% initialize arrays for data saving
Tt = zeros(non,ceil(not/param.save.intervall));
s = 1;
if param.convection == 1
    Ut = zeros(nx-1,ny,ceil(not/param.save.intervall));
    Vt = zeros(nx,ny-1,ceil(not/param.save.intervall));
end

% initialize nodal temperature matrix
T_new = zeros(non,1);
T_old = T_init(:);
T_0 = zeros(not,1);

% initialize staggered grid velocity matrices
U_old = zeros(nx-1,ny);
V_old = zeros(nx,ny-1);
U_new = zeros(nx-1,ny);
V_new = zeros(nx,ny-1);

% add boundaries to the velocity matriced
```

```
58  Uwb = [zeros(1,ny); U_old(:,:) ;zeros(1,ny)];   % east and west boundaries
59  Uwb = [-Uwb(:,1) Uwb -Uwb(:,end)];               % north and south boundaries
60  Vwb = [zeros(nx,1) V_old(:,:) zeros(nx,1)];      % north and south boundaries
61  Vwb = [-Vwb(1,:); Vwb ;-Vwb(end,:)];             % east and west boundaries
62  Ue = avg(Uwb')';
63  Ve = avg(Vwb);
64
65  % initialize solid parts
66  elempcm = elem_mat(reshape(1:nel,nx,ny)) == 1;
67  T_ij = reshape(T_old(:),nx+1,ny+1);      % staggered grid temperatures
68  %Tl = param.pcm.Tm+param.pcm.dTm;         % temperature of complete
        liquidification
69  Tl = param.pcm.Tm-param.pcm.dTm;          % temperature of partial
        liquidification
70  % p_el = phase of element [0 = solid, 1 = liquid]
71  p_el = false(nx,ny);
72
73  % initialize heat flow and enthalpy arrays
74  Q_in = zeros(not,1);
75  Q_out = zeros(not,1);
76  H = zeros(not,1);
77  H_l = zeros(not,1);
78
79  %% FEM preprocessing
80  % calculate shape matrices
81  [gauss] = shape_mat(param);
82  % preprocess FEM data
83  [gauss] = FEM_preprocessor(T_old,gauss,param);
84
85  %% time stepping
86  for t = 1:not
87
88      %% FEM calculation
89      [C,K,R,Qin_e,Qout_e,H(t),H_l(t)] = ...
90              heat2Delem_new(t,T_old,Uwb,Vwb,p_el,gauss,param);
91
92      % heat flows and enthalpy
93      %%{
94      if t > 1
95          Q_in(t) = Q_in(t-1) + Qin_e;          % input heat flow
96          Q_out(t) = Q_out(t-1) + Qout_e;          % output heat flow
97      else
98          Q_in(t) = Qin_e;                      % input heat flow
99          Q_out(t) = Qout_e;                    % output heat flow
100     end
101     %}
102
103     %% solve equation for temperatures at t+1
104     T_new = solve_euler_back(T_old,C,K,R,param);
105
106     % refresh staggered grid temperatures
107     T_ij = reshape(T_new,nx+1,ny+1); % staggered grid temperatures
108     if param.convection == 1
109         p_el = min(min( T_ij(1:end-1,1:ny),T_ij(1:end-1,2:ny+1)),...
110                     min(T_ij(2:end,1:ny),T_ij(2:end,2:ny+1) ))>Tl...
111                     & elempcm;
112     end
113
114     %% solve velocities at t+1
115     if any(any(p_el)) == 1
116
117         %% operating temperature for boussinesq term
118         % mean temperature in liquid region
119         T_0(t) = mean(sum(T_new(IEN(:,reshape(p_el,[],1)==1)))/4);
120         % fixed temperature
121         %T_0(t) = 230;
122         param.pcm.T_0 = T_0(t);
123
124         %% solve navier stokes equations
125         [U_new,V_new] = solve_navierstokes(U_old,V_old,T_ij,p_el,param);
126
127         %% reset velocities in solid part
128         U_new = U_new.*(p_el(1:nx-1,:) & p_el(2:nx,:));
129         V_new = V_new.*(p_el(:,1:ny-1) & p_el(:,2:ny));
130
131         %% add boundary values to velocitiy matrices
132         % in domain phase boundary
133         U_new(:,1:end-1) = U_new(:,1:end-1)-U_new(:,2:end).*...
134             (p_el(1:end-1,2:end)==1 & p_el(1:end-1,2:end)~=p_el(1:end-1,1:end
                -1));
```

```
135         U_new(:,2:end) = U_new(:,2:end)-U_new(:,1:end-1).*...
136             (p_el(1:end-1,1:end-1)==1 & p_el(1:end-1,1:end-1)~=p_el(1:end
                    -1,2:end));
137         V_new(1:end-1,:) = V_new(1:end-1,:)-V_new(2:end,:).*...
138             (p_el(2:end,1:end-1)==1 & p_el(2:end,1:end-1)~=p_el(1:end-1,1:end
                    -1));
139         V_new(2:end,:) = V_new(2:end,:)-V_new(1:end-1,:).*...
140             (p_el(1:end-1,1:end-1)==1 & p_el(1:end-1,1:end-1)~=p_el(2:end,1:
                    end-1));
141
142         % domain boundaries
143         Uwb = [zeros(1,ny+2); [-U_new(:,1) U_new(:,:) -U_new(:,end)];...
144                zeros(1,ny+2)];
145         Vwb = [[0 -V_new(1,:) 0] ;[zeros(nx,1) V_new(:,:) zeros(nx,1)];...
146                [0 -V_new(end,:) 0]];
147
148     else
149         U_new = U_old;
150         V_new = V_old;
151     end
152
153     %% waitbar update
154     if param.display.waitbar == 1
155         [f,timearray,up] = waitbar2D_simple(f,param,timearray,toc,t,up);
156     end
157     %% dump variables
158     if mod(t,param.save.intervall) == 0
159         Tt(:,s) = T_new;
160         if param.convection == 1
161             Ut(:,:,s) = U_new;
162             Vt(:,:,s) = V_new;
163         end
164         s = s+1;
165     end
166     if param.save.make_snapshot == 1
167         if mod(t,param.save.snapshot_int)==0
168             strFilename = sprintf('results/dump_%s_t%03d.mat',param.save.name
                        ,t);
169             if param.convection == 1
170                 save(strFilename,'C','K','R','t','T_new','T_old','U_new',...
171                 'V_new','U_old','V_old','H','Q_in','Q_out','H_l','param');
172             else
173                 save(strFilename,'C','K','R','t','T_new','T_old',...
174                                  'H','Q_in','Q_out','H_l','param');
175             end
176         end
177     end
178
179     %% initialize T,U,V for next timestep
180     T_old = T_new;
181     U_old = U_new;
182     V_old = V_new;
183 end
184
185 %% postprocessing
186 % delete waitbar
187 if param.display.waitbar == 1
188     delete(f);
189 end
190 % save data
191 strFilename = sprintf('results/dump_%s_t%03d_final.mat',param.save.name,t);
192 save(strFilename);
193
194 % plots after total time
195 if param.display.Tplot == 1
196     makeplot(T_new,param);
197 end
198 if param.display.Vplot == 1
199 makeplot_konv(U_new,V_new,param);
200 end
201
202 % show simulation time
203 strTimestamp = datestr(now(),30); % get timestamp
204 fprintf('Simulation complete / timestamp %s\n',strTimestamp);
205 sim_time = toc;
206 fprintf('Total simulation time [s]: %5.0f\n',toc);
207 end
```

Code Listing A.2: Code of function mesh2d

```
%% creation of the 2D mesh
function [param] = mesh2d(param)
% The 2D mesh is created. Coordinates and connectivities of nodes are
% computed. Boundary elements and element materials are flagged.
% All information is written to the structure array param.

%% load input data
nx = param.mesh.nx;
ny = param.mesh.ny;
non = param.mesh.non;
nel = param.mesh.nel;
v2struct(param.geom);

%% generate coordinates of nodes
x0 = linspace(0,x_length,nx+1);      % equal bisection of the x nodes
y0 = linspace(0,y_length,ny+1);      % equal bisection of the y nodes
x = zeros(1,non);
y = zeros(1,non);
for i = 1:(ny+1)
    for j = 1:(nx+1)
        x( (i-1)*(nx+1)+j ) = x0(j);
        y( (i-1)*(nx+1)+j ) = y0(i);
    end
end
param.mesh.x = x;
param.mesh.y = y;

%% generate the IEN connectivity array
% IEN = identify element nodes
IEN = zeros(4,nel);
rowcount = 0;
for elementcount = 1:nel
    IEN(1,elementcount) = elementcount + rowcount;
    IEN(2,elementcount) = elementcount + 1 + rowcount;
    IEN(3,elementcount) = elementcount + (nx + 2) + rowcount;
    IEN(4,elementcount) = elementcount + (nx + 1) + rowcount;
    if mod(elementcount,nx) == 0
        rowcount = rowcount + 1;
    end
end
param.mesh.IEN = IEN;

%% flag boundary elements
flag = zeros(nel,2);
for e = 1:nel
    if mod(e,nx) == 1
        flag(e,1) = 3;        % 3 = left side element
    elseif mod(e,nx) == 0
        flag(e,1) = 1;        % 1 = right side element
    end
    for i = 1:nx
        flag(i,2) = 4;        % 4 = lower side element
        flag((nel-i+1),2) = 2;  % 2 = upper side element
    end
end
param.mesh.flag = flag;

%% flag element matieral
% coordinates of element center
center_coord = zeros(2,nel);
center_coord(:,:) = [(x(IEN(2,:))+x(IEN(1,:)))/2;(y(IEN(3,:))+y(IEN(2,:)))
    /2];

% flag element matieral
elem_mat = zeros(nel,1);
for i = length(x_length_vec):-1:1
    for j = length(y_length_vec):-1:1
        x = sum(x_length_vec(1:i));
        y = sum(y_length_vec(1:j));
        elem_mat(center_coord(1,:)<x & center_coord(2,:)<y) = mat_mtrx(j,i);
    end
end
param.mesh.elem_mat = elem_mat;
param.mesh.center_coord = center_coord;

%% generate staggered grid coordinates for convection
stag = zeros(param.mesh.nel,2);
for e = 1:param.mesh.nel
```

```
    if mod(e,param.mesh.nx) == 0
        stag(e,1) = param.mesh.nx;
    else
        stag(e,1)  = mod(e,param.mesh.nx);
    end
    stag(e,2) = ceil(e/param.mesh.nx);
end
param.mesh.stag = stag;
```

Code Listing A.3: Code of function shape_mat

```
%% shape function calculation
function [gauss] = shape_mat(param)
% If all elements are the same size, use this function to calculate the
% shape functions, derivate of shape functions and Jacobian to be later
% used in the FEM Calculation heat2Delem_2.

nodes = param.mesh.IEN(:,1);
coord  = [param.mesh.x(nodes); param.mesh.y(nodes)]';
gp = [-0.57735027, 0.57735027 ];    % gauss points (2 Point Gauss-Legendre)

% for element terms
gauss.N1 = NmatHeat2D(gp(1),gp(1));
gauss.N1_q = gauss.N1'*gauss.N1;
[gauss.B1, gauss.detJ] = BmatHeat2D(gp(1),gp(1),coord);
gauss.B1_q = gauss.B1'*gauss.B1;

gauss.N2 = NmatHeat2D(gp(2),gp(1));
gauss.N2_q = gauss.N2'*gauss.N2;
gauss.B2 = BmatHeat2D(gp(2),gp(1),coord);
gauss.B2_q = gauss.B2'*gauss.B2;

gauss.N3 = NmatHeat2D(gp(2),gp(2));
gauss.N3_q = gauss.N3'*gauss.N3;
gauss.B3 = BmatHeat2D(gp(2),gp(2),coord);
gauss.B3_q = gauss.B3'*gauss.B3;

gauss.N4 = NmatHeat2D(gp(1),gp(2));
gauss.N4_q = gauss.N4'*gauss.N4;
gauss.B4 = BmatHeat2D(gp(1),gp(2),coord);
gauss.B4_q = gauss.B4'*gauss.B4;

% for boundary terms
eta = 1;
gauss.Ne1 = NmatHeat2D(eta,gp(1));
gauss.Ne1_q = gauss.Ne1'*gauss.Ne1;
gauss.Ne2 = NmatHeat2D(eta,gp(2));
gauss.Ne2_q = gauss.Ne2'*gauss.Ne2;
gauss.dS_e = 0.25*[eta-1  -eta-1  1+eta    1-eta]*coord(:,2);
eta = -1;
gauss.Nw1 = NmatHeat2D(eta,gp(1));
gauss.Nw1_q = gauss.Nw1'*gauss.Nw1;
gauss.Nw2 = NmatHeat2D(eta,gp(2));
gauss.Nw2_q = gauss.Nw2'*gauss.Nw2;
gauss.dS_w = - 0.25*[eta-1  -eta-1  1+eta    1-eta]*coord(:,2);

% for convection terms
gp = [-1,0,1];          % convection gauss points
w = [1/3,3/4,1/3];      % convection gauss weights
gauss.Nc1 = NmatHeat2D(gp(1),gp(1));        % eta -1, psi -1
[B1, detJ] = BmatHeat2D(gp(1),gp(1),coord);
mat_const = param.pcm.rho*param.pcm.c_l*detJ;
gauss.NBc1u = w(1)*w(1)*mat_const*gauss.Nc1'*B1(1,:);
gauss.NBc1v = w(1)*w(1)*mat_const*gauss.Nc1'*B1(2,:);

gauss.Nc2 = NmatHeat2D(gp(1),gp(2));        % eta -1, psi 0
B2 = BmatHeat2D(gp(1),gp(2),coord);
gauss.NBc2u = w(1)*w(2)*mat_const*gauss.Nc2'*B2(1,:);
gauss.NBc2v = w(1)*w(2)*mat_const*gauss.Nc2'*B2(2,:);

gauss.Nc3 = NmatHeat2D(gp(1),gp(3));        % eta -1, psi 1
B3 = BmatHeat2D(gp(1),gp(3),coord);
gauss.NBc3u = w(1)*w(3)*mat_const*gauss.Nc3'*B3(1,:);
gauss.NBc3v = w(1)*w(3)*mat_const*gauss.Nc3'*B3(2,:);

gauss.Nc4 = NmatHeat2D(gp(2),gp(1));        % eta  0, psi -1
B4 = BmatHeat2D(gp(2),gp(1),coord);
```

```
gauss.NBc4u = w(2)*w(1)*mat_const*gauss.Nc4'*B4(1,:);
gauss.NBc4v = w(2)*w(1)*mat_const*gauss.Nc4'*B4(2,:);

gauss.Nc5 = NmatHeat2D(gp(2),gp(2));        % eta  0, psi 0
B5 = BmatHeat2D(gp(2),gp(2),coord);
gauss.NBc5u = w(2)*w(2)*mat_const*gauss.Nc5'*B5(1,:);
gauss.NBc5v = w(2)*w(2)*mat_const*gauss.Nc5'*B5(2,:);

gauss.Nc6 = NmatHeat2D(gp(2),gp(3));        % eta  0, psi 1
B6 = BmatHeat2D(gp(2),gp(3),coord);
gauss.NBc6u = w(2)*w(3)*mat_const*gauss.Nc6'*B6(1,:);
gauss.NBc6v = w(2)*w(3)*mat_const*gauss.Nc6'*B6(2,:);

gauss.Nc7 = NmatHeat2D(gp(3),gp(1));        % eta  1, psi -1
B7 = BmatHeat2D(gp(3),gp(1),coord);
gauss.NBc7u = w(3)*w(1)*mat_const*gauss.Nc7'*B7(1,:);
gauss.NBc7v = w(3)*w(1)*mat_const*gauss.Nc7'*B7(2,:);

gauss.Nc8 = NmatHeat2D(gp(3),gp(2));        % eta  1, psi 0
B8 = BmatHeat2D(gp(3),gp(2),coord);
gauss.NBc8u = w(3)*w(2)*mat_const*gauss.Nc8'*B8(1,:);
gauss.NBc8v = w(3)*w(2)*mat_const*gauss.Nc8'*B8(2,:);

gauss.Nc9 = NmatHeat2D(gp(3),gp(3));        % eta  1, psi 1
B9 = BmatHeat2D(gp(3),gp(3),coord);
gauss.NBc9u = w(3)*w(3)*mat_const*gauss.Nc9'*B9(1,:);
gauss.NBc9v = w(3)*w(3)*mat_const*gauss.Nc9'*B9(2,:);

end
```

Code Listing A.4: Code of shape matrix function NmatHeat2D

```
%% N matrix Heat2D
function N = NmatHeat2D(eta,psi)
% Computes shape functions matrix N

N  = 0.25 * [(1-psi)*(1-eta)  (1-psi)*(1+eta)  ...
             (1+psi)*(1+eta)  (1+psi)*(1-eta)]; % shape functions
```

Code Listing A.5: Code of shape matrix function BmatHeat2D

```
%% B matrix Heat2D
function [B, detJ] = BmatHeat2D(eta,psi,coord)
% Computes B matrix
% = derivative of the shape functions matrix N

% Calculate the Grad(N) matrix
GN    = 0.25 * [psi-1  1-psi   1+psi   -psi-1;
                eta-1  -eta-1  1+eta    1-eta];

J     = GN*coord;   % compute Jacobian matrix
detJ  = det(J);     % Jacobian

B     = J\GN;       % compute the B matrix
```

Code Listing A.6: Code of FEM preprocessing function FEM_preprocessor

```
%% FEM preprocessor
function [gauss] = FEM_preprocessor(T_old,gauss,param)
% preprocesses element properties for FEM calculation

% load important variables
elem_mat = param.mesh.elem_mat;
IEN = param.mesh.IEN;
nel = param.mesh.nel;
elem_pcm = find(elem_mat==1); % element indices of PCM

%% intialize preprocessed data arrays
c1 = zeros(nel,1);
c2=c1;c3=c1;c4=c1;k1=c1;k2=c1;k3=c1;k4=c1;rho=c1;
flag_el=c1;calc_h=c1;
```

```
15
16  %% preprocess data for 4 gauss points per element
17
18  % heat capacities and heat conductivities
19  % gauss point 1
20  T1 = gauss.N1*T_old(IEN(:,:));
21  c1(elem_mat==1) = c_var(T1(elem_mat==1),param.pcm);
22  k1(elem_mat==1) = k_var(T1(elem_mat==1),param.pcm);
23  for m = 2:size(param.geom.material,1)
24      str1 = strcat('param.',param.geom.material{m,2},'.c_s');
25          str2 = strcat('param.',param.geom.material{m,2},'.k_s');
26      c1(elem_mat==m) = eval(str1);
27          k1(elem_mat==m) = eval(str2);
28  end
29  % gauss point 2
30  T2 = gauss.N2*T_old(IEN(:,:));
31  c2(elem_mat==1) = c_var(T2(elem_mat==1),param.pcm);
32  k2(elem_mat==1) = k_var(T2(elem_mat==1),param.pcm);
33  for m = 2:size(param.geom.material,1)
34      str1 = strcat('param.',param.geom.material{m,2},'.c_s');
35          str2 = strcat('param.',param.geom.material{m,2},'.k_s');
36      c2(elem_mat==m) = eval(str1);
37          k2(elem_mat==m) = eval(str2);
38  end
39  % gauss point 3
40  T3 = gauss.N3*T_old(IEN(:,:));
41  c3(elem_mat==1) = c_var(T3(elem_mat==1),param.pcm);
42  k3(elem_mat==1) = k_var(T3(elem_mat==1),param.pcm);
43  for m = 2:size(param.geom.material,1)
44      str1 = strcat('param.',param.geom.material{m,2},'.c_s');
45          str2 = strcat('param.',param.geom.material{m,2},'.k_s');
46      c3(elem_mat==m) = eval(str1);
47          k3(elem_mat==m) = eval(str2);
48  end
49  % gauss point 4
50  T4 = gauss.N4*T_old(IEN(:,:));
51  c4(elem_mat==1) = c_var(T4(elem_mat==1),param.pcm);
52  k4(elem_mat==1) = k_var(T4(elem_mat==1),param.pcm);
53  for m = 2:size(param.geom.material,1)
54      str1 = strcat('param.',param.geom.material{m,2},'.c_s');
55          str2 = strcat('param.',param.geom.material{m,2},'.k_s');
56      c4(elem_mat==m) = eval(str1);
57          k4(elem_mat==m) = eval(str2);
58  end
59
60  % density of element
61  for m = 1:size(param.geom.material,1)
62      str = strcat('param.',param.geom.material{m,2},'.rho');
63      rho(elem_mat==m) = eval(str);
64  end
65
66  % constants for enthalpy calculation
67  for m = 1:size(param.geom.material,1)
68      str1 = strcat('param.',param.geom.material{m,2},'.rho');
69      str2 = strcat('param.',param.geom.material{m,2},'.c_s');
70      calc_h(elem_mat==m) = eval(str1)*gauss.detJ*eval(str2);
71  end
72
73  %% preprocess boundary information
74  % flag boundary elements by one identifier
75  flag_el(param.mesh.flag(:,1) == 3) = 3; % left side element
76  flag_el(param.mesh.flag(:,1) == 1) = 1; % right side element
77  flag_el(param.mesh.flag(:,2) == 2) = 2; % upper side element
78  flag_el(param.mesh.flag(:,2) == 4) = 4; % lower side element
79  flag_el((param.mesh.flag(:,1) == 3)&(param.mesh.flag(:,2) == 4) ) = 34;
80  flag_el((param.mesh.flag(:,1) == 3)&(param.mesh.flag(:,2) == 2) ) = 32;
81  flag_el((param.mesh.flag(:,1) == 1)&(param.mesh.flag(:,2) == 4) ) = 14;
82  flag_el((param.mesh.flag(:,1) == 1)&(param.mesh.flag(:,2) == 2) ) = 12;
83
84  gauss.west = find(param.mesh.flag(:,1) == 3);
85  gauss.east = find(param.mesh.flag(:,1) == 1);
86  yr=param.mesh.y(param.mesh.IEN(:,:));
87  yr1 = yr(1,:)+(yr(4,:)-yr(1,:))*(1-0.57735027)/2;
88  yr2 = yr(1,:)+(yr(4,:)-yr(1,:))*(1+0.57735027)/2;
89  gauss.Tout1 = @(t) param.icbc.Tout(t,yr1);
90  gauss.Tout2 = @(t) param.icbc.Tout(t,yr2);
91  gauss.alpha_out1 = @(t) param.icbc.alpha_out(t,yr1);
92  gauss.alpha_out2 = @(t) param.icbc.alpha_out(t,yr2);
93
94  yl=param.mesh.y(param.mesh.IEN(:,:));
```

```
yl1 = yl(1,:)+(yl(4,:)-yl(1,:))*(1-0.57735027)/2;
yl2 = yl(1,:)+(yl(4,:)-yl(1,:))*(1+0.57735027)/2;
gauss.Tin1 = @(t) param.icbc.Tin(t,yl1);
gauss.Tin2 = @(t) param.icbc.Tin(t,yl2);
gauss.alpha_in1 = @(t) param.icbc.alpha_in(t,yl1);
gauss.alpha_in2 = @(t) param.icbc.alpha_in(t,yl2);

%% preprocess sparse assembling relation
sp_mat_e = cell(nel,1);
sp_vec_e = cell(nel,1);
for e = 1:nel
    je = param.mesh.IEN(:,e);

    k = [je;je;je;je];
    l = [je(1);je(1);je(1);je(1);...
         je(2);je(2);je(2);je(2);...
         je(3);je(3);je(3);je(3);...
         je(4);je(4);je(4);je(4)];

    sp_mat_e{e} = [k,l];
    sp_vec_e{e} = [je,[1;1;1;1]];
end
sp_mat = cell2mat(sp_mat_e);
sp_vec = cell2mat(sp_vec_e);

%% load variables to struct variable gauss
gauss.rho = rho;
gauss.elem_pcm = elem_pcm;
gauss.sp_mat = sp_mat;
gauss.sp_vec = sp_vec;
gauss.flag_el = flag_el;
gauss.calc_h = calc_h;
gauss.c1 = c1;
gauss.c2 = c2;
gauss.c3 = c3;
gauss.c4 = c4;
gauss.k1 = k1;
gauss.k2 = k2;
gauss.k3 = k3;
gauss.k4 = k4;

end
```

Code Listing A.7: Code of FEM matrix assembly function heat2Delem_new

```
%% heat 2D element new
function [C,K,R,Qin,Qout,H,H_l] = heat2Delem_new(t,T,U,V,p_el,gauss,param)
% Computes the finite element matrices

%% initialize variables and matrix cells
nel = param.mesh.nel;
non = param.mesh.non;
nx = param.mesh.nx;
elem_pcm = gauss.elem_pcm;
not_pcm = param.mesh.elem_mat ~= 1;
sp_mat = gauss.sp_mat;
sp_vec = gauss.sp_vec;
flag_el = gauss.flag_el;
calc_h = gauss.calc_h;
calc_h1 = gauss.calc_h;
calc_h2 = gauss.calc_h;
calc_h3 = gauss.calc_h;
calc_h4 = gauss.calc_h;
hl_1 = gauss.calc_h;
hl_2 = gauss.calc_h;
hl_3 = gauss.calc_h;
hl_4 = gauss.calc_h;
c1 = gauss.c1;
c2 = gauss.c2;
c3 = gauss.c3;
c4 = gauss.c4;
k1 = gauss.k1;
k2 = gauss.k2;
k3 = gauss.k3;
k4 = gauss.k4;

p_el_FEM = reshape(p_el,[],1);
%C_e = cell(nel,1);
```

```
%K_e = cell(nel,1);
%R_e = cell(nel,1);

%% vectorized preprocessing of element data
% temperatures at gauss points
T1 = gauss.N1*T(param.mesh.IEN(:,:));
T2 = gauss.N2*T(param.mesh.IEN(:,:));
T3 = gauss.N3*T(param.mesh.IEN(:,:));
T4 = gauss.N4*T(param.mesh.IEN(:,:));
Te1 = gauss.Ne1*T(param.mesh.IEN(:,:));
Te2 = gauss.Ne2*T(param.mesh.IEN(:,:));
Tw1 = gauss.Nw1*T(param.mesh.IEN(:,:));
Tw2 = gauss.Nw2*T(param.mesh.IEN(:,:));

% for conductive terms
% heat capacities and heat conductivities at gauss points of pcm elements
c1(elem_pcm) = c_var(T1(elem_pcm),param.pcm);
k1(elem_pcm) = k_var(T1(elem_pcm),param.pcm);
c2(elem_pcm) = c_var(T2(elem_pcm),param.pcm);
k2(elem_pcm) = k_var(T2(elem_pcm),param.pcm);
c3(elem_pcm) = c_var(T3(elem_pcm),param.pcm);
k3(elem_pcm) = k_var(T3(elem_pcm),param.pcm);
c4(elem_pcm) = c_var(T4(elem_pcm),param.pcm);
k4(elem_pcm) = k_var(T4(elem_pcm),param.pcm);
% constants for enthalpy calculation at gauss points of pcm elements
[calc_h1(elem_pcm),hl_1(elem_pcm)] = h_var(T1(elem_pcm),param,gauss.detJ);
[calc_h2(elem_pcm),hl_2(elem_pcm)] = h_var(T2(elem_pcm),param,gauss.detJ);
[calc_h3(elem_pcm),hl_3(elem_pcm)] = h_var(T3(elem_pcm),param,gauss.detJ);
[calc_h4(elem_pcm),hl_4(elem_pcm)] = h_var(T4(elem_pcm),param,gauss.detJ);

% for convective terms
% velocities at gauss points of pcm elements
if any(p_el_FEM) == 1
        ie = param.mesh.stag(:,1);
        je = param.mesh.stag(:,2);
        % eta -1, psi -1
        u1 = (U(sub2ind(size(U),ie,je))+U(sub2ind(size(U),ie,je+1)))/2;
        v1 = (V(sub2ind(size(V),ie,je))+V(sub2ind(size(V),ie+1,je)))/2;
        % eta -1, psi 0
        u2 = U(sub2ind(size(U),ie,je+1));
        v2 = (V(sub2ind(size(V),ie,je))+V(sub2ind(size(V),ie,je+1))+...
          V(sub2ind(size(V),ie+1,je))+V(sub2ind(size(V),ie+1,je+1)) )/2;
        % eta -1, psi 1
        u3 = (U(sub2ind(size(U),ie,je+1))+U(sub2ind(size(U),ie,je+2)))/2;
        v3 = (V(sub2ind(size(V),ie,je+1))+V(sub2ind(size(V),ie+1,je+1)))/2;
        % eta  0, psi -1
        u4 = (U(sub2ind(size(U),ie,je))+U(sub2ind(size(U),ie,je+1))+...
          U(sub2ind(size(U),ie+1,je))+U(sub2ind(size(U),ie+1,je+1)) )/2;
        v4 = V(sub2ind(size(V),ie,je+1));
        % eta  0, psi 0
        u5 = (U(sub2ind(size(U),ie,je+1))+U(sub2ind(size(U),ie+1,je+1)))/2;
        v5 = (V(sub2ind(size(V),ie+1,je))+V(sub2ind(size(V),ie+1,je+1)))/2;
        % eta  0, psi 1
        u6 = (U(sub2ind(size(U),ie,je+1))+U(sub2ind(size(U),ie,je+2))+...
          U(sub2ind(size(U),ie+1,je+1))+U(sub2ind(size(U),ie+1,je+2)) )/2;
        v6 = V(sub2ind(size(V),ie+1,je+1));
        % eta  1, psi -1
        u7 = (U(sub2ind(size(U),ie+1,je))+U(sub2ind(size(U),ie+1,je+1)))/2;
        v7 = (V(sub2ind(size(V),ie+1,je))+V(sub2ind(size(V),ie+2,je)))/2;
        % eta  1, psi 0
        u8 = U(sub2ind(size(U),ie+1,je+1));
        v8 = (V(sub2ind(size(V),ie+1,je))+V(sub2ind(size(V),ie+1,je+1))+...
          V(sub2ind(size(V),ie+2,je))+V(sub2ind(size(V),ie+2,je+1)) )/2;
        % eta  1, psi 1
        u9 = (U(sub2ind(size(U),ie+1,je+1))+U(sub2ind(size(U),ie+1,je+2)))/2;
        v9 = (V(sub2ind(size(V),ie+1,je+1))+V(sub2ind(size(V),ie+2,je+1)))/2;

        %% velocity correction
    % mark elements for velocity correction
        correct_u = zeros(nel,1);
        correct_v = zeros(nel,1);
        solid=p_el_FEM==0;

        % middle element
        correct_u(flag_el==0) = solid(circshift(flag_el==0,1))*1+...
                solid(circshift(flag_el==0,-1))*(-1)+...
              ( solid(circshift(flag_el==0,1)) & ...
                solid(circshift(flag_el==0,-1)))*2;
        correct_v(flag_el==0) = solid(circshift(flag_el==0,nx))*1+...
                solid(circshift(flag_el==0,-nx))*(-1)+...
```

```
              ( solid(circshift(flag_el==0,nx)) & ...
                solid(circshift(flag_el==0,-nx)) )*2;
        % left side element
        correct_u(flag_el==3) = -1+solid(circshift(flag_el==3,1))*3;
        correct_v(flag_el==3) = solid(circshift(flag_el==3,nx))*1+...
                solid(circshift(flag_el==3,-nx))*(-1)+...
              ( solid(circshift(flag_el==3,nx)) & ...
                solid(circshift(flag_el==3,-nx)) )*2;
        correct_u(flag_el==32) = -1+solid(circshift(flag_el==32,1))*3;
        correct_v(flag_el==32) = 1+solid(circshift(flag_el==32,-nx))*1;
        correct_u(flag_el==34) = -1+solid(circshift(flag_el==34,1))*3;
        correct_v(flag_el==34) = -1+solid(circshift(flag_el==32,nx))*3;
        % right side element
        correct_u(flag_el==1) = 1+solid(circshift(flag_el==1,-1))*1;
        correct_v(flag_el==1) = solid(circshift(flag_el==1,nx))*1+...
                solid(circshift(flag_el==1,-nx))*(-1)+...
              ( solid(circshift(flag_el==1,nx)) & ...
                solid(circshift(flag_el==1,-nx)) )*2;
        correct_u(flag_el==12) = 1+solid(circshift(flag_el==12,-1))*1;
        correct_v(flag_el==12) = 1+solid(circshift(flag_el==12,-nx))*1;
        correct_u(flag_el==14) = 1+solid(circshift(flag_el==14,-1))*1;
        correct_v(flag_el==14) = -1+solid(circshift(flag_el==14,nx))*3;
        % upper side element
        correct_u(flag_el==2) = solid(circshift(flag_el==2,1))*1+...
                solid(circshift(flag_el==2,-1))*(-1)+...
              ( solid(circshift(flag_el==2,1)) & ...
                solid(circshift(flag_el==2,-1)) )*2;
        correct_v(flag_el==2) = 1+solid(circshift(flag_el==2,-nx))*1;
        % lower side element
        correct_u(flag_el==4) = solid(circshift(flag_el==4,1))*1+...
                solid(circshift(flag_el==4,-1))*(-1)+...
              ( solid(circshift(flag_el==4,1)) & ...
                solid(circshift(flag_el==4,-1)) )*2;
        correct_v(flag_el==4) = -1+solid(circshift(flag_el==4,nx))*3;

    % perform velocity correction
        u=[u1,u2,u3,u4,u5,u6,u7,u8,u9];
    v=[v1,v2,v3,v4,v5,v6,v7,v8,v9];
        u(correct_u==2,:) = 0;
        u(correct_u==1,:) = max(0,u(correct_u==1,:));
        u(correct_u==-1,:) = min(0,u(correct_u==-1,:));
        v(correct_v==2,:) = 0;
        v(correct_v==1,:) = max(0,v(correct_v==1,:));
        v(correct_v==-1,:) = min(0,v(correct_v==-1,:));
        u1 = u(:,1); u2 = u(:,2); u3 = u(:,3); u4 = u(:,4); u5 = u(:,5);
        u6 = u(:,6); u7 = u(:,7); u8 = u(:,8); u9 = u(:,9);
        v1 = v(:,1); v2 = v(:,2); v3 = v(:,3); v4 = v(:,4); v5 = v(:,5);
        v6 = v(:,6); v7 = v(:,7); v8 = v(:,8); v9 = v(:,9);
end

%% calculate element matrices
Ce2 = gauss.detJ*(...
        bsxfun(@times,gauss.N1_q,permute(gauss.rho'.*c1',[3 1 2]))+...
        bsxfun(@times,gauss.N2_q,permute(gauss.rho'.*c2',[3 1 2]))+...
        bsxfun(@times,gauss.N3_q,permute(gauss.rho'.*c3',[3 1 2]))+...
        bsxfun(@times,gauss.N4_q,permute(gauss.rho'.*c4',[3 1 2])));
C_e = reshape(Ce2,4*4*nel,1);

KCe = gauss.detJ*(...
        bsxfun(@times,gauss.B1_q,permute(k1',[3 1 2]))+...
        bsxfun(@times,gauss.B2_q,permute(k2',[3 1 2]))+...
        bsxfun(@times,gauss.B3_q,permute(k3',[3 1 2]))+...
        bsxfun(@times,gauss.B4_q,permute(k4',[3 1 2])));

right = param.mesh.flag(:,1) == 1; % right side of PCM
KB1e = gauss.dS_e*( ...
        bsxfun(@times,gauss.Ne1_q,permute(right'.*gauss.alpha_out1(t),[3 1
            2]))+...
        bsxfun(@times,gauss.Ne2_q,permute(right'.*gauss.alpha_out2(t),[3 1
            2])));

RB1e = gauss.dS_e*( ...
        bsxfun(@times,gauss.Ne1',permute(right'.*gauss.alpha_out1(t).*gauss.
            Tout1(t),[3 1 2]))+...
        bsxfun(@times,gauss.Ne2',permute(right'.*gauss.alpha_out2(t).*gauss.
            Tout2(t),[3 1 2])));

left = param.mesh.flag(:,1) == 3; % left side of PCM
KB3e = -gauss.dS_w*( ...
```

```
189         bsxfun(@times,gauss.Nw1_q,permute(left'.*gauss.alpha_in1(t),[3 1 2]))
                +...
190         bsxfun(@times,gauss.Nw2_q,permute(left'.*gauss.alpha_in2(t),[3 1 2]))
                );
191
192 RB3e = -gauss.dS_w*( ...
193         bsxfun(@times,gauss.Nw1',permute(left'.*gauss.alpha_in1(t).*gauss.
                Tin1(t),[3 1 2]))+...
194         bsxfun(@times,gauss.Nw2',permute(left'.*gauss.alpha_in2(t).*gauss.
                Tin2(t),[3 1 2])));
195
196 % convection term
197 if any(p_el_FEM) == 1
198     KCe = KCe + bsxfun(@times,gauss.NBc1u,permute(p_el_FEM'.*u1',[3 1 2]))+
            ...
199                 bsxfun(@times,gauss.NBc2u,permute(p_el_FEM'.*u2',[3 1 2]))+
                        ...
200                 bsxfun(@times,gauss.NBc3u,permute(p_el_FEM'.*u3',[3 1 2]))+
                        ...
201                 bsxfun(@times,gauss.NBc4u,permute(p_el_FEM'.*u4',[3 1 2]))+
                        ...
202                 bsxfun(@times,gauss.NBc5u,permute(p_el_FEM'.*u5',[3 1 2]))+
                        ...
203                 bsxfun(@times,gauss.NBc6u,permute(p_el_FEM'.*u6',[3 1 2]))+
                        ...
204                 bsxfun(@times,gauss.NBc7u,permute(p_el_FEM'.*u7',[3 1 2]))+
                        ...
205                 bsxfun(@times,gauss.NBc8u,permute(p_el_FEM'.*u8',[3 1 2]))+
                        ...
206                 bsxfun(@times,gauss.NBc9u,permute(p_el_FEM'.*u9',[3 1 2]))+
                        ...
207                 bsxfun(@times,gauss.NBc1v,permute(p_el_FEM'.*v1',[3 1 2]))+
                        ...
208                 bsxfun(@times,gauss.NBc2v,permute(p_el_FEM'.*v2',[3 1 2]))+
                        ...
209                 bsxfun(@times,gauss.NBc3v,permute(p_el_FEM'.*v3',[3 1 2]))+
                        ...
210                 bsxfun(@times,gauss.NBc4v,permute(p_el_FEM'.*v4',[3 1 2]))+
                        ...
211                 bsxfun(@times,gauss.NBc5v,permute(p_el_FEM'.*v5',[3 1 2]))+
                        ...
212                 bsxfun(@times,gauss.NBc6v,permute(p_el_FEM'.*v6',[3 1 2]))+
                        ...
213                 bsxfun(@times,gauss.NBc7v,permute(p_el_FEM'.*v7',[3 1 2]))+
                        ...
214                 bsxfun(@times,gauss.NBc8v,permute(p_el_FEM'.*v8',[3 1 2]))+
                        ...
215                 bsxfun(@times,gauss.NBc9v,permute(p_el_FEM'.*v9',[3 1 2]));
216 end
217 K_e = reshape(KCe + KB1e + KB3e,4*4*nel,1);
218 R_e = reshape(RB1e + RB3e,4*nel,1);
219
220 %% assemble to global matrices and vectors
221 C = sparse(sp_mat(:,1),sp_mat(:,2),C_e,non,non);
222 K = sparse(sp_mat(:,1),sp_mat(:,2),K_e,non,non);
223 R = sparse(sp_vec(:,1),sp_vec(:,2),R_e,non,1);
224
225 %% calculate enthalpy values
226 H = sum(    calc_h1(elem_pcm)+calc_h2(elem_pcm)+...
227             calc_h3(elem_pcm)+calc_h4(elem_pcm)) + ...
228     sum(    calc_h(not_pcm)'.*(T1(not_pcm)+...
229             T2(not_pcm)+T3(not_pcm)+T4(not_pcm)));
230
231 H_l = sum(  hl_1(elem_pcm)+hl_2(elem_pcm)+hl_3(elem_pcm)+hl_4(elem_pcm) );
232
233 Qout = gauss.dS_e*param.mesh.d_t*sum(...
234         gauss.alpha_out1(t).*(gauss.Tout1(t) - Te1).*right' +...
235         gauss.alpha_out2(t).*(gauss.Tout2(t) - Te2).*right');
236
237 Qin = -gauss.dS_w*param.mesh.d_t*sum(...
238         gauss.alpha_in1(t).*(gauss.Tin1(t) - Tw1).*left' +...
239         gauss.alpha_in2(t).*(gauss.Tin2(t) - Tw2).*left');
240
241 %old version
242 %{
243 for e = 1:nel
244         KB1e  = zeros(4,4);
245         KB3e  = zeros(4,4);
246         RB1e  = zeros(4,1);
247         RB3e  = zeros(4,1);
```

```
        %Ce    = rho(e)*gauss.detJ*(c1(e).*gauss.N1_q +...
    %            c2(e).*gauss.N2_q +c3(e).*gauss.N3_q +...
    %            c4(e).*gauss.N4_q);
        KCe   = gauss.detJ*(k1(e).*gauss.B1_q + ...
                k2(e).*gauss.B2_q + k3(e).*gauss.B3_q +...
                k4(e).*gauss.B4_q );

    % calculate enthaply
    if param.mesh.elem_mat(e) == 1
        H{e} =  calc_h1(e)+calc_h2(e)+calc_h3(e)+calc_h4(e);
        H_l{e} = hl_1(e)+hl_2(e)+hl_3(e)+hl_4(e);
    else
        H{e} =  calc_h(e)*(T1(e)+T2(e)+T3(e)+T4(e));
    end

        if param.mesh.flag(e,1) == 1 % right side of PCM
        nodes = param.mesh.IEN(:,e);
        coord  = [param.mesh.x(nodes); param.mesh.y(nodes)]';
        y1 = coord(1,2)+(coord(4,2)-coord(1,2))*(-0.57735027/2);
        y2 = coord(1,2)+(coord(4,2)-coord(1,2))*(0.57735027/2);

        KB1e  = param.icbc.alpha_out*gauss.dS_e*( ...
                gauss.Ne1_q + gauss.Ne2_q);

        RB1e =  param.icbc.alpha_out*gauss.dS_e*...
               (param.icbc.Tout(t,y1)*gauss.Ne1'+...
                param.icbc.Tout(t,y2)*gauss.Ne2');

        % calculate heat flow
        Qout{e} = gauss.dS_e*param.icbc.alpha_out*param.mesh.d_t*(...
                param.icbc.Tout(t,y1) - Te1(e) + ...
                param.icbc.Tout(t,y2) - Te2(e));

        elseif param.mesh.flag(e,1) == 3 % left side of PCM
        nodes = param.mesh.IEN(:,e);
        coord  = [param.mesh.x(nodes); param.mesh.y(nodes)]';
        y1 = coord(1,2)+(coord(4,2)-coord(1,2))*(-0.57735027/2);
        y2 = coord(1,2)+(coord(4,2)-coord(1,2))*(0.57735027/2);

        KB3e = -gauss.dS_w*(param.icbc.alpha_in(t,y1)*gauss.Nw1_q...
                + param.icbc.alpha_in(t,y2)*gauss.Nw2_q);

        RB3e = -gauss.dS_w*(param.icbc.alpha_in(t,y1)*...
                param.icbc.Tin(t,y1)*gauss.Nw1'+...
                param.icbc.alpha_in(t,y2)*param.icbc.Tin(t,y2)...
                *gauss.Nw2');

        % calculate heat flow
        Qin{e} = -gauss.dS_w*param.mesh.d_t*(...
                  param.icbc.alpha_in(t,y1)*(param.icbc.Tin(t,y1)-Tw1(e))...
                  +param.icbc.alpha_in(t,y2)*(param.icbc.Tin(t,y2)-Tw2(e)));
    end

        % convection term
        if p_el_FEM(e) == 1

        KCe = KCe + u1(e)*gauss.NBc1u+v1(e)*gauss.NBc1v+...
                    u2(e)*gauss.NBc2u+v2(e)*gauss.NBc2v+...
                    u3(e)*gauss.NBc3u+v3(e)*gauss.NBc3v+...
                    u4(e)*gauss.NBc4u+v4(e)*gauss.NBc4v+...
                    u5(e)*gauss.NBc5u+v5(e)*gauss.NBc5v+...
                    u6(e)*gauss.NBc6u+v6(e)*gauss.NBc6v+...
                    u7(e)*gauss.NBc7u+v7(e)*gauss.NBc7v+...
                    u8(e)*gauss.NBc8u+v8(e)*gauss.NBc8v+...
                    u9(e)*gauss.NBc9u+v9(e)*gauss.NBc9v;
        end
        K_e{e} = reshape(KCe + KB1e + KB3e,4*4,1);
        R_e{e} = RB1e + RB3e ;
        C_e{e} = reshape(Ce,4*4,1);
end

%% assemble to global matrices and vectors

%C = sparse(sp_mat(:,1),sp_mat(:,2),cell2mat(C_e),non,non);
%K = sparse(sp_mat(:,1),sp_mat(:,2),cell2mat(K_e),non,non);
%R = sparse(sp_vec(:,1),sp_vec(:,2),cell2mat(R_e),non,1);
%}
end
```

Code Listing A.8: Code of function c_var

```
%% c_var
function [c]=c_var(T,mat)
% Computes heat capacity c from the value of temperature T

c = mat.c_s*( T <= (mat.Tm - mat.dTm) )+...
    mat.c_l*( T >= (mat.Tm + mat.dTm) )+...
    (mat.lh + mat.c_s*(mat.Tm + mat.dTm - T) + ...
        mat.c_l*(T-mat.Tm + mat.dTm))/(2*mat.dTm).*...
    ( T > (mat.Tm - mat.dTm) & T < (mat.Tm + mat.dTm) );
end
```

Code Listing A.9: Code of function k_var

```
%% k_var
function [k]=k_var(T,mat)
% Computes heat conductivity k from the value of temperature T

k = mat.k_s*( T <= (mat.Tm - mat.dTm) )+...
    mat.k_l*( T >= (mat.Tm + mat.dTm) )+...
    (mat.k_s*(mat.Tm + mat.dTm - T) + ...
        mat.k_l*(T-mat.Tm + mat.dTm))/(2*mat.dTm).*...
    ( T > (mat.Tm - mat.dTm) & T < (mat.Tm + mat.dTm) );
end
```

Code Listing A.10: Code of function solve_euler_back

```
%% solve euler back
function [Tnew] = solve_euler_back(Told,C,K,R,param)
% Solves the system of FEM equations via the backwards euler method

Tnew = Told +param.mesh.d_t*...
       ( (C + param.mesh.d_t*K)\(R-K*Told) );
% long version:
%{
% build equation system A*X=b
M_eff = C + param.mesh.d_t*K;
b = R-K*Told(:);

% solve equation system A*X=b
X = M_eff\b;
Tnew = Told +param.mesh.d_t*X;
%}
```

Code Listing A.11: Code of function waitbar2D_simple

```
%% waitbar 2D simple
function [f,timearray,up] = waitbar2D_simple(f,param,timearray,toc,t,up)
% updates waitbar in a simple way

wait = t/param.mesh.not;
timearray(t) = toc-sum(timearray);
% moving average of computation time
average = movmean(timearray,[20 0],'Endpoints',timearray(end));
waittime = (param.mesh.not-t)*average(end);

if (toc-up) > param.display.update
        % wait at least a specified time between updates
        up = toc;

        if waittime < 60
            waitbar(wait,f,sprintf(...
          'Fortschritt: %2.1f %%,  Restdauer: ~ %2.1f s',100*wait,waittime));
    else
        waittime_s = mod(waittime,60);
        waittime_m = floor(waittime/60);
        waitbar(wait,f,sprintf(...
        'Fortschritt: %2.1f %%,  Restdauer: ~ %3.0f min %2.0f s',...
        100*wait,waittime_m,waittime_s));
        end
end
end
```

Code Listing A.12: Code of function solve_navierstokes

```
%% solve navier stokes
function [U_new,V_new,grad,U,V] = solve_navierstokes(U,V,T_ij,p_el,param)
% Solves the navier stokes equations via finite difference method

%% preprocessing
nx = param.mesh.nx;
ny = param.mesh.ny;
f_u = reshape((p_el(1:nx-1,:) & p_el(2:nx,:)),[],1);
f_v = reshape((p_el(:,1:ny-1) & p_el(:,2:ny)),[],1);
% temperatures for boussinesque approximation
T_v = reshape(avg(T_ij(:,2:end-1)),[],1);
T_u = reshape(avg(T_ij(2:end-1,:)')',[],1);

f2_v = 0+(T_v>=param.pcm.Tm+param.pcm.dTm).*1+...
       (T_v>param.pcm.Tm-param.pcm.dTm & T_v<param.pcm.Tm+param.pcm.dTm).*...
        (T_v-param.pcm.Tm+param.pcm.dTm)/(2*param.pcm.dTm);
f2_u = 0+(T_u>=param.pcm.Tm+param.pcm.dTm).*1+...
       (T_u>param.pcm.Tm-param.pcm.dTm & T_u<param.pcm.Tm+param.pcm.dTm).*...
        (T_u-param.pcm.Tm+param.pcm.dTm)/(2*param.pcm.dTm);

%% nonlinear terms / explicit time step
gamma = min(1.2*param.mesh.d_t*max( ...
    max(max(abs(U)))/param.geom.dx,max(max(abs(V)))/param.geom.dy ),1);
% add boundary conditions
Ue = [zeros(1,ny+2); [-U(:,1) U -U(:,end)] ;zeros(1,ny+2)];
Ve = [0 -V(1,:) 0; [zeros(nx,1) V zeros(nx,1)];0 -V(end,:) 0];

% upwinding
Ua = avg(Ue')';                     % vertical average of U
Ud = diff(Ue')'/2;                  % vertical finite difference of U
Va = avg(Ve);                       % horizontal average of V
Vd = diff(Ve)/2;                    % horizontal finite difference of V
UVx = diff(Ua.*Va-gamma*abs(Ua).*Vd)/param.geom.dx;     % d(UV)/dx
UVy = diff((Ua.*Va-gamma*Ud.*abs(Va))')'/param.geom.dy; % d(UV)/dy
Ua = avg(Ue(:,2:end-1));            % horizontal average of U
Ud = diff(Ue(:,2:end-1))/2;         % horizontal finite difference of U
Va = avg(Ve(2:end-1,:)')';          % vertical average of V
Vd = diff(Ve(2:end-1,:)')'/2;       % vertical finite difference of V
U2x = diff(Ua.^2-gamma*abs(Ua).*Ud)/param.geom.dx;      % d(UU)/dx
V2y = diff((Va.^2-gamma*abs(Va).*Vd)')'/param.geom.dy;  % d(VV)/dx
% explicit time step
U = U-param.mesh.d_t*(UVy(2:end-1,:)+U2x);
V = V-param.mesh.d_t*(UVx(:,2:end-1)+V2y);

%% viscosity terms / implicit time step
% implicit step for U
Ex=sparse(2:nx-1,1:nx-2,1,nx-1,nx-1);
Ax=Ex+Ex'-2*speye(nx-1);
Ey=sparse(2:ny,1:ny-1,1,ny,ny);
Ay=Ey+Ey'-2*speye(ny);
Ay(1,1)=-3; Ay(ny,ny)=-3;       %Neumann B.Cs
A=kron(Ay/param.geom.dy^2,speye(nx-1))+kron(speye(ny),Ax/param.geom.dx^2);

Uo=reshape(U,[],1);
mue = param.pcm.mue*(1+(1-f2_u).^4*10^3);
Fu = (1-f_u).^2./(f_u.^3+0.001)*10^8;
Un = (speye(length(Uo)).*(1+Fu) - param.mesh.d_t/param.pcm.rho*mue.*A) \ ...
     (Uo - sin(param.pcm.phi)*param.mesh.d_t*param.pcm.grav*...
     param.pcm.beta*(param.pcm.T_0 - T_u));
U=reshape(Un,nx-1,ny);

% implicit step for V
Ex=sparse(2:nx,1:nx-1,1,nx,nx);
Ax=Ex+Ex'-2*speye(nx);
Ax(1,1)=-3; Ax(nx,nx)=-3;       %Neumann B.Cs
Ey=sparse(2:ny-1,1:ny-2,1,ny-1,ny-1);
Ay=Ey+Ey'-2*speye(ny-1);
A=kron(Ay/param.geom.dy^2,speye(nx))+kron(speye(ny-1),Ax/param.geom.dx^2);

Vo=reshape(V,[],1);
mue = param.pcm.mue*(1+(1-f2_v).^4*10^3);
Fv = (1-f_v).^2./(f_v.^3+0.001)*10^8;
Vn = (speye(length(Vo)).*(1+Fv) - param.mesh.d_t/param.pcm.rho*mue.*A) \ ...
     (Vo - cos(param.pcm.phi)*param.mesh.d_t*param.pcm.grav*...
     param.pcm.beta*(param.pcm.T_0 - T_v));
V=reshape(Vn,nx,ny-1);

```

```
%% pressure correction
grad = reshape(diff([zeros(1,ny);U;zeros(1,ny)])/param.geom.dx...
        +diff([zeros(nx,1) V zeros(nx,1)]')'/param.geom.dy,[],1);

Lp_p = Laplace_p(p_el,param);
p = Lp_p\(param.pcm.rho/param.mesh.d_t*grad);

P = reshape(p,nx,ny);
U_new = U-param.mesh.d_t/param.pcm.rho*diff(P)/param.geom.dx;
V_new = V-param.mesh.d_t/param.pcm.rho*diff(P')'/param.geom.dy;

end
```

Code Listing A.13: Code of function Laplace_p

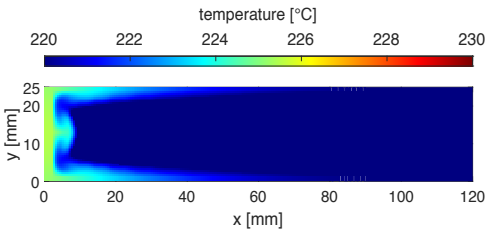
```
%% Laplace p
function [Lp_p] = Laplace_p(p_el,param)
% Computes the finite difference 2D Laplace stencil for the pressure p.
% Includes homogeneous Neumann boundary conditions on the internal phase
% boundaries.

dxq = param.geom.dx^2;
dyq = param.geom.dy^2;
nx = param.mesh.nx;
ny = param.mesh.ny;

ps = reshape([false(nx,1) p_el(:,1:end-1)],[],1);
pn = reshape([p_el(:,2:end) false(nx,1)],[],1);
pw = reshape([false(1,ny); p_el(1:end-1,:)],[],1);
pe = reshape([p_el(2:end,:); false(1,ny)],[],1);
p_el = reshape(p_el,[],1);

i = param.mesh.stag(:,1);
j = param.mesh.stag(:,2);

Lp_val = [i+(j-1)*nx, i+(j-1)*nx,...
          0.1-1/dxq*(p_el==pw)-1/dxq*(p_el==pe)-1/dyq*(p_el==ps)-1/dyq*(p_el
              ==pn);
          i+(j-1)*nx, i-1+(j-1)*nx, 0+1/dxq*(p_el==pw);
          i+(j-1)*nx, i+1+(j-1)*nx, 0+1/dxq*(p_el==pe);
          i+(j-1)*nx, i+(j-2)*nx,   0+1/dyq*(p_el==ps);
          i+(j-1)*nx, i+(j)*nx,     0+1/dyq*(p_el==pn);
          ];
      % small Value 0.1, so that Lp_p is not singular.
Lp_val = Lp_val(find((Lp_val(:,2)>0)&(Lp_val(:,2)<=param.mesh.nel)),:);

Lp_p = sparse(Lp_val(:,1),Lp_val(:,2),Lp_val(:,3),param.mesh.nel,param.mesh.
    nel);

end
```

Appendix B

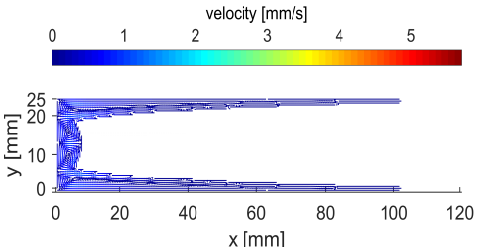
Additional plots

To illustrate the simulation results of the parameter study discussed in Section 6.2, additional plots which show the temperature and velocity distribution in the simulated cavities at six time points during the respective simulations are presented here.

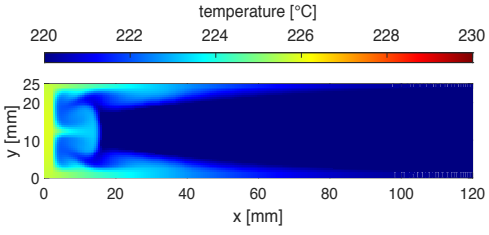
For the sake of improving comparability of the given plots, the illustrations begin on the following double page, so that each double page contains the temperature and velocity distributions of a particular cavity orientation case.



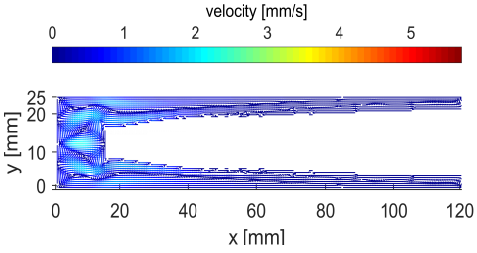
(a) temperature after 22 minutes



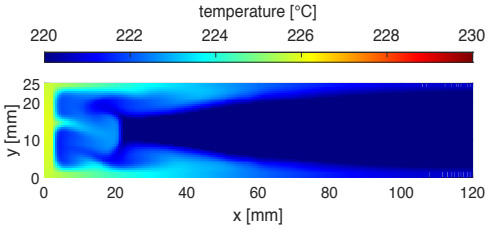
(b) velocity after 22 minutes



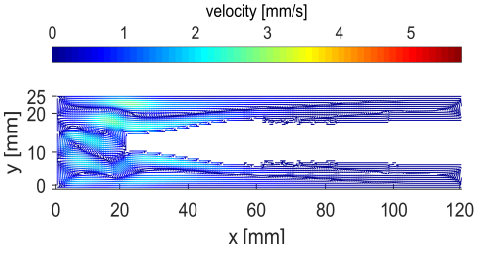
(c) temperature after 45 minutes



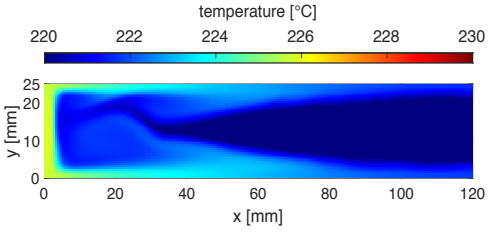
(d) velocity after 45 minutes



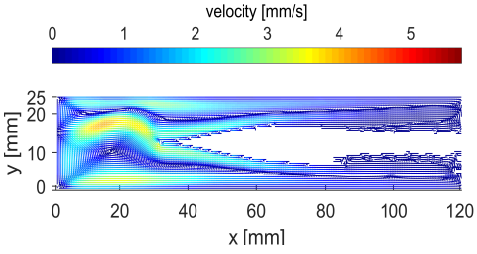
(e) temperature after 67 minutes



(f) velocity after 67 minutes

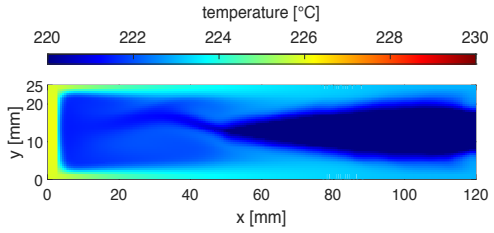


(g) temperature after 90 minutes

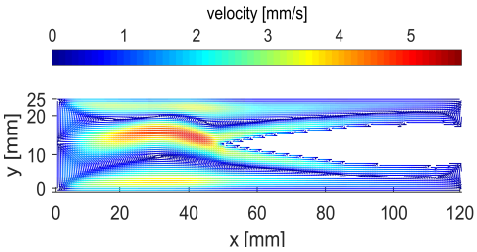


(h) velocity after 90 minutes

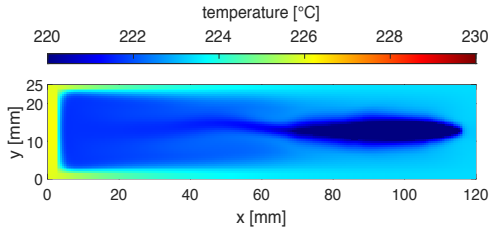
Figure B.1: Temperature and velocity distribution at different simulation times for cavity orientation 0°



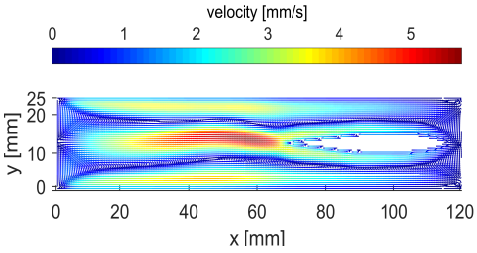
(i) temperature after 112 minutes



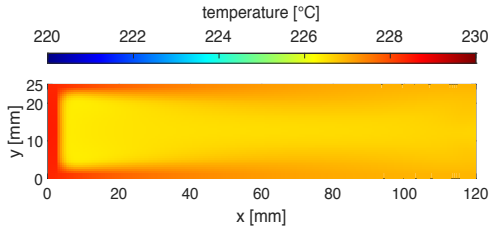
(j) velocity after 112 minutes



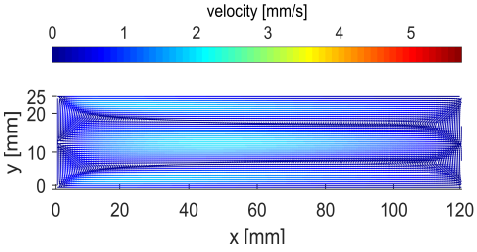
(k) temperature after 135 minutes



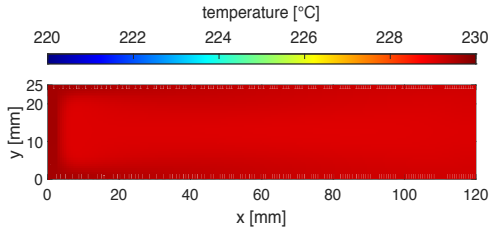
(l) velocity after 135 minutes



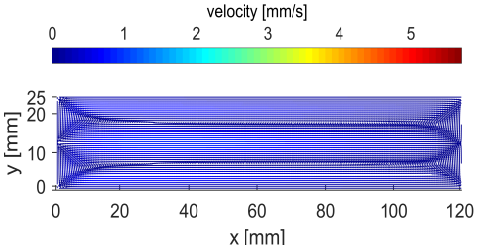
(m) temperature after 157 minutes



(n) velocity after 157 minutes

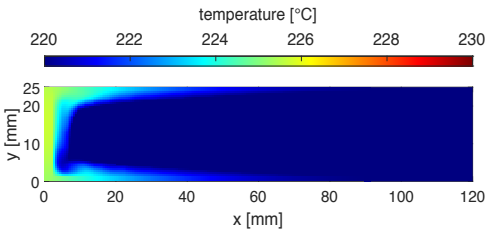


(o) temperature after 180 minutes

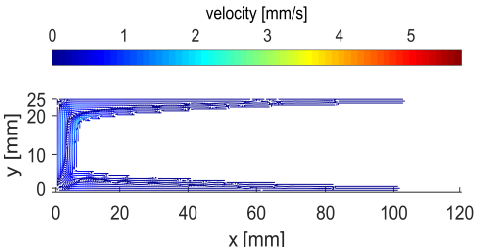


(p) velocity after 180 minutes

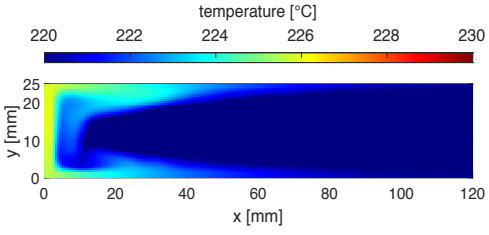
Figure B.1: Temperature and velocity distribution at different simulation times for cavity orientation 0°



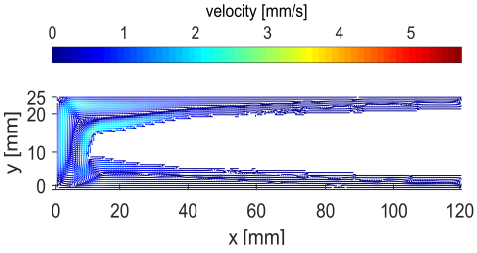
(q) temperature after 22 minutes



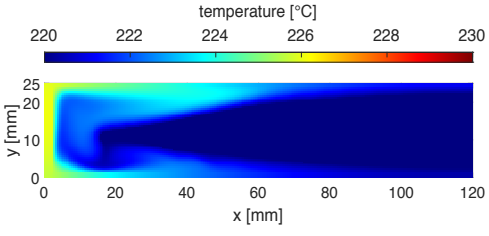
(r) velocity after 22 minutes



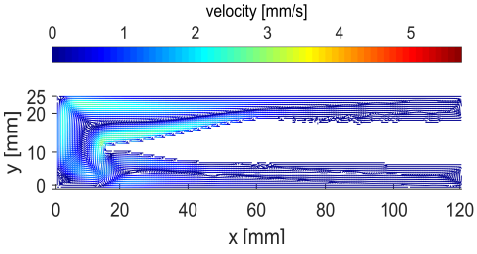
(s) temperature after 45 minutes



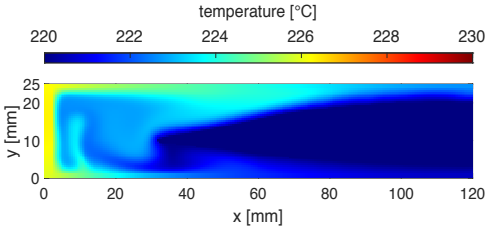
(t) velocity after 45 minutes



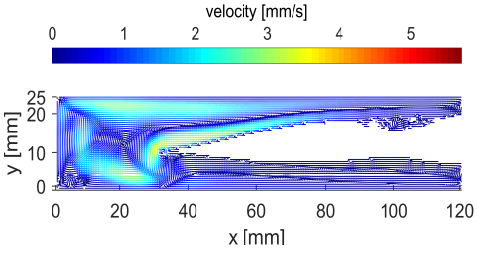
(u) temperature after 67 minutes



(v) velocity after 67 minutes

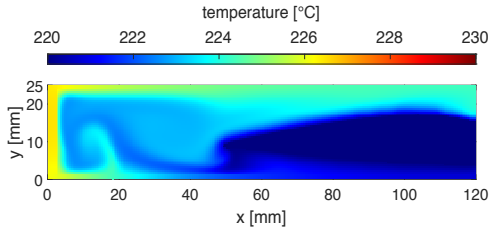


(w) temperature after 90 minutes

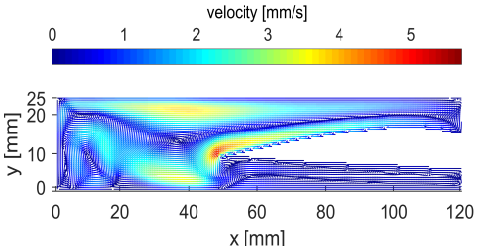


(x) velocity after 90 minutes

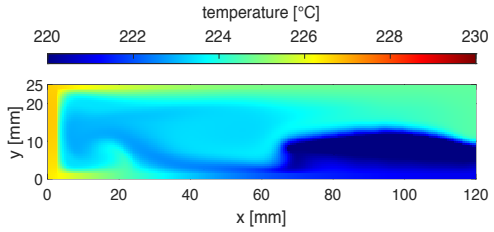
Figure B.2: Temperature and velocity distribution at different simulation times for cavity orientation 45°



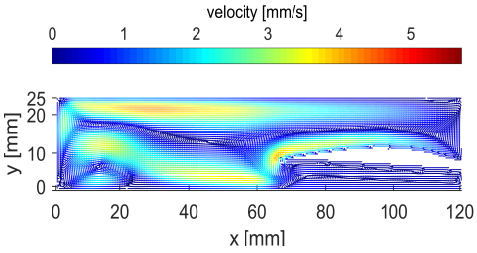
(i) temperature after 112 minutes



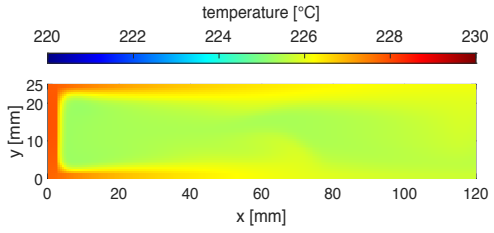
(j) velocity after 112 minutes



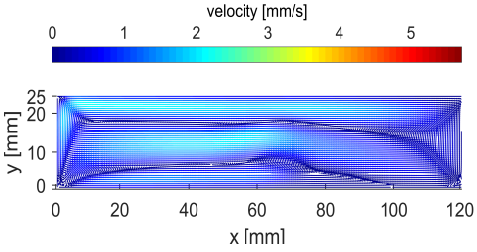
(k) temperature after 135 minutes



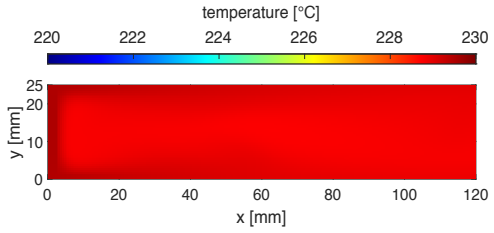
(l) velocity after 135 minutes



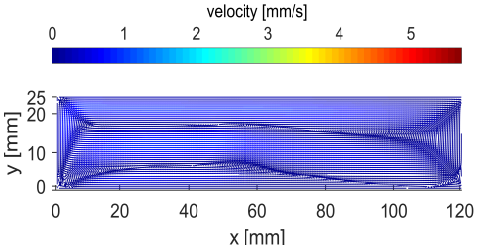
(m) temperature after 157 minutes



(n) velocity after 157 minutes

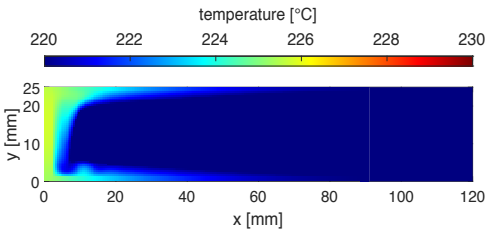


(o) temperature after 180 minutes

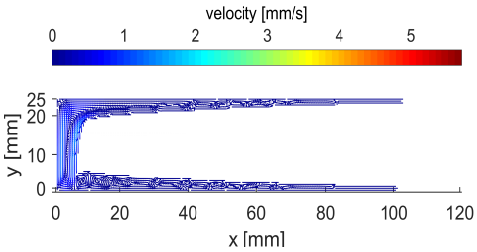


(p) velocity after 180 minutes

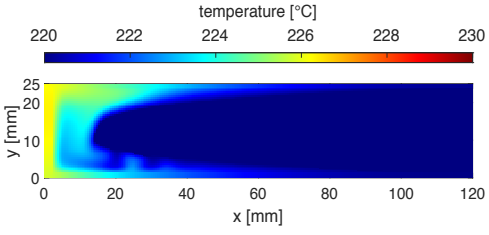
Figure B.2: Temperature and velocity distribution at different simulation times for cavity orientation 45°



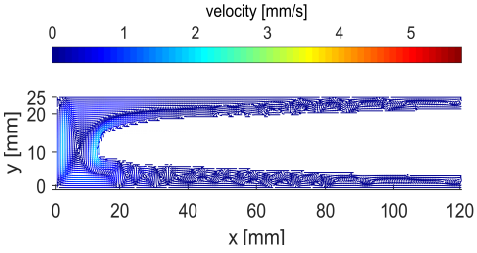
(q) temperature after 22 minutes



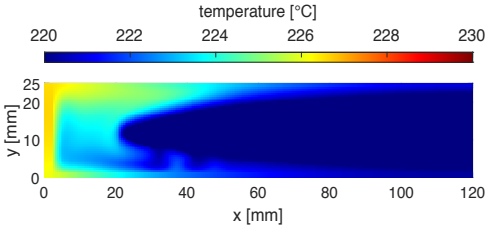
(r) velocity after 22 minutes



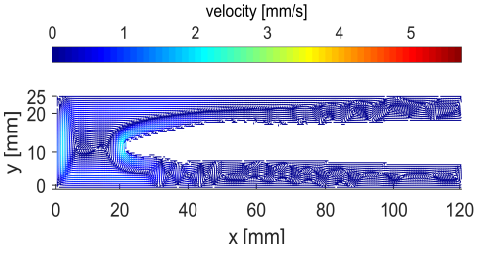
(s) temperature after 45 minutes



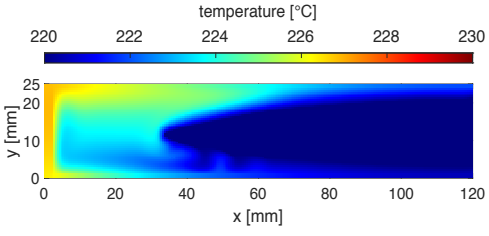
(t) velocity after 45 minutes



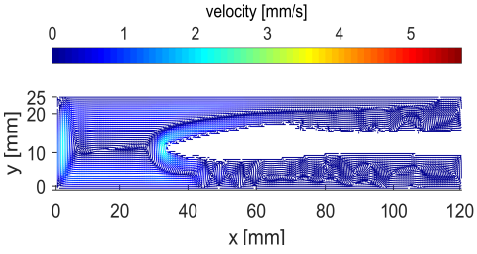
(u) temperature after 67 minutes



(v) velocity after 67 minutes

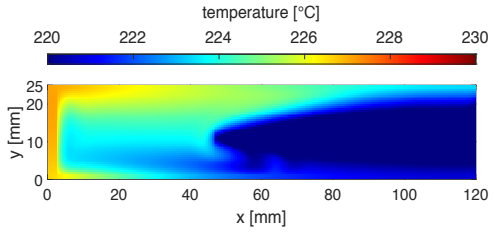


(w) temperature after 90 minutes

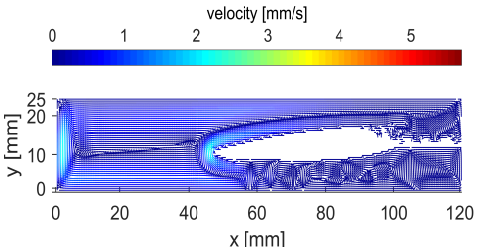


(x) velocity after 90 minutes

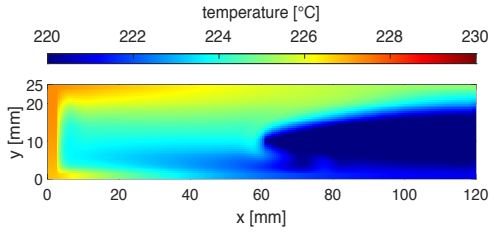
Figure B.3: Temperature and velocity distribution at different simulation times for cavity orientation 90°



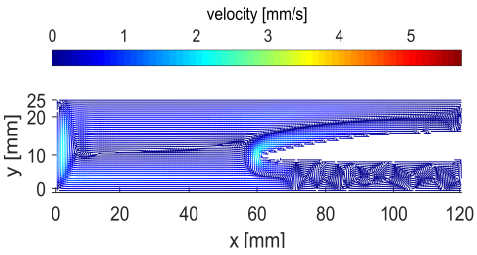
(i) temperature after 112 minutes



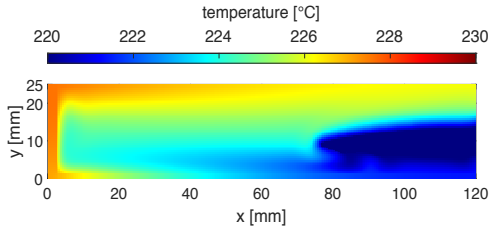
(j) velocity after 112 minutes



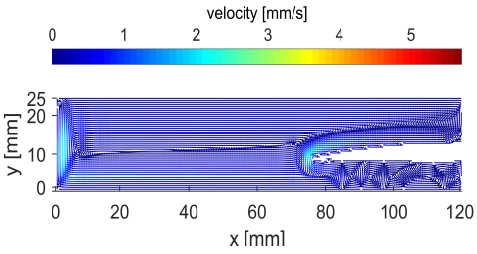
(k) temperature after 135 minutes



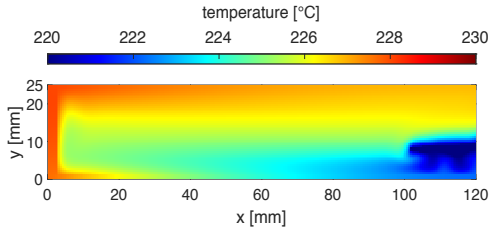
(l) velocity after 135 minutes



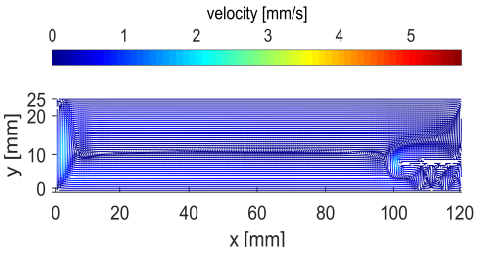
(m) temperature after 157 minutes



(n) velocity after 157 minutes

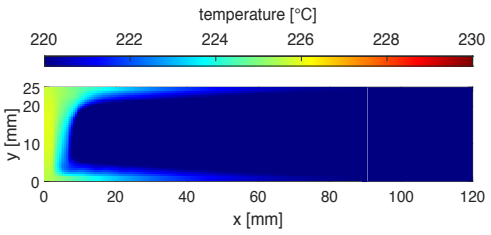


(o) temperature after 180 minutes

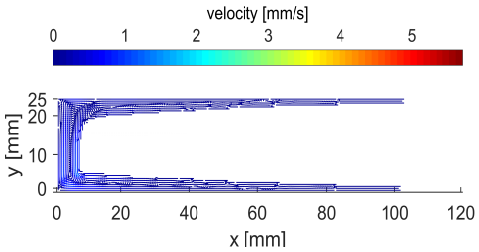


(p) velocity after 180 minutes

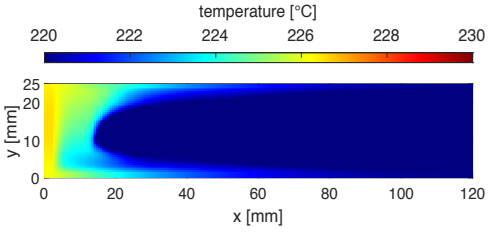
Figure B.3: Temperature and velocity distribution at different simulation times for cavity orientation 90°



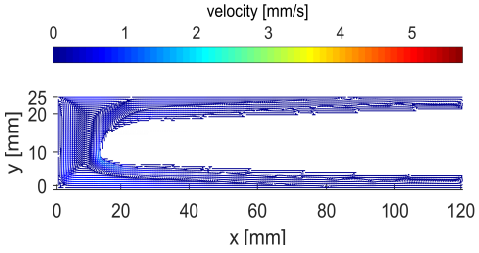
(q) temperature after 22 minutes



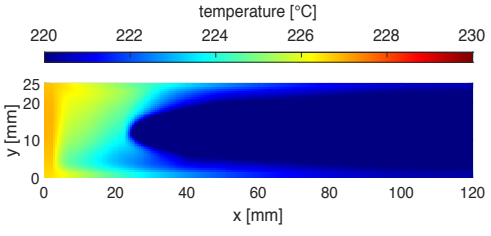
(r) velocity after 22 minutes



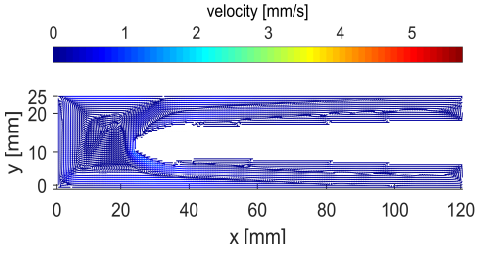
(s) temperature after 45 minutes



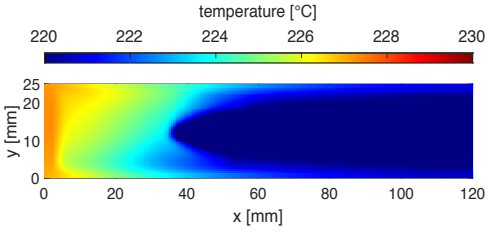
(t) velocity after 45 minutes



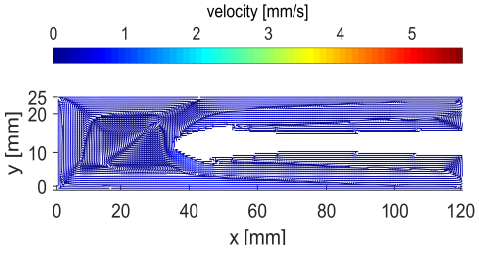
(u) temperature after 67 minutes



(v) velocity after 67 minutes

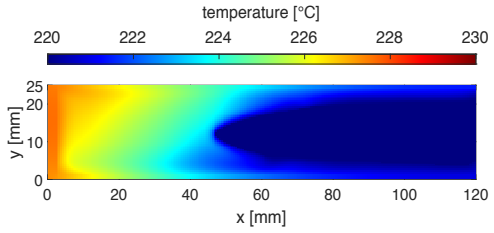


(w) temperature after 90 minutes

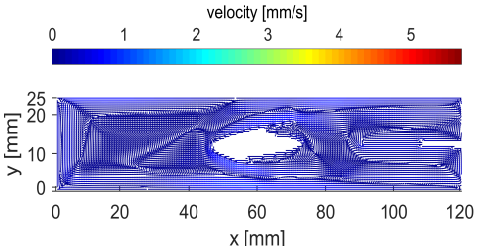


(x) velocity after 90 minutes

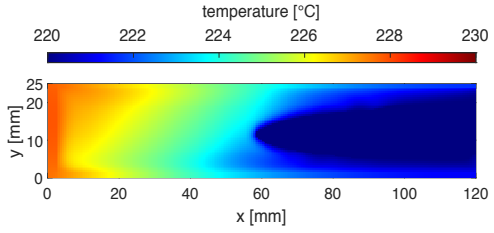
Figure B.4: Temperature and velocity distribution at different simulation times for cavity orientation 135°



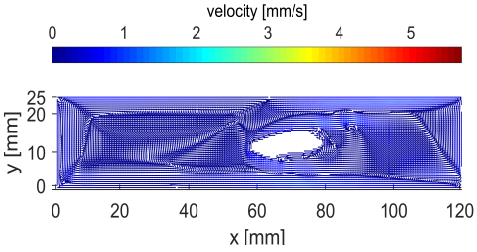
(i) temperature after 112 minutes



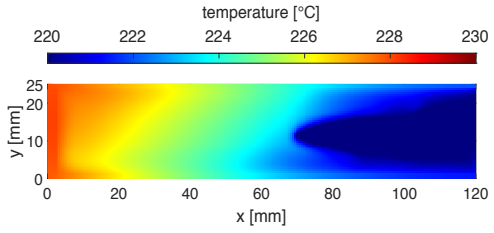
(j) velocity after 112 minutes



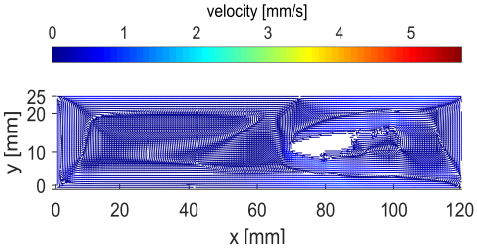
(k) temperature after 135 minutes



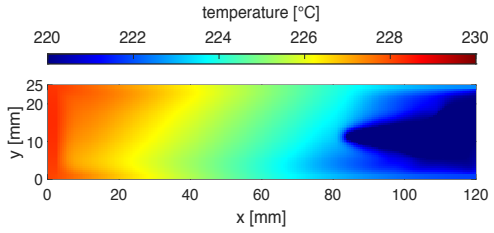
(l) velocity after 135 minutes



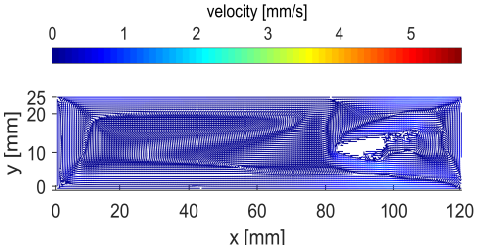
(m) temperature after 157 minutes



(n) velocity after 157 minutes

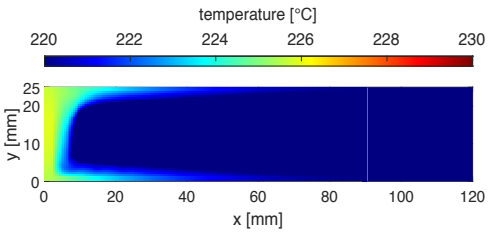


(o) temperature after 180 minutes

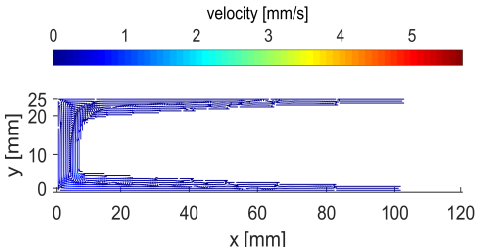


(p) velocity after 180 minutes

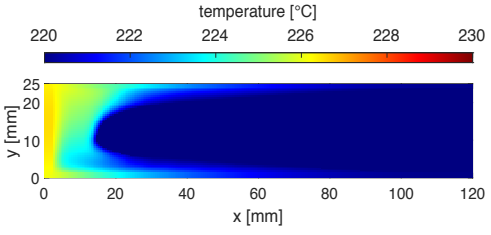
Figure B.4: Temperature and velocity distribution at different simulation times for cavity orientation 135°



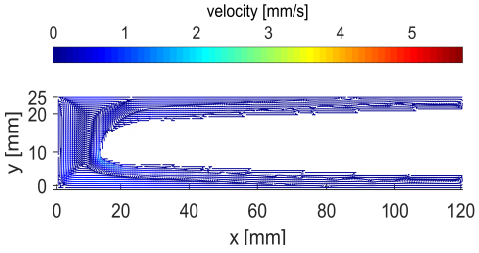
(q) temperature after 22 minutes



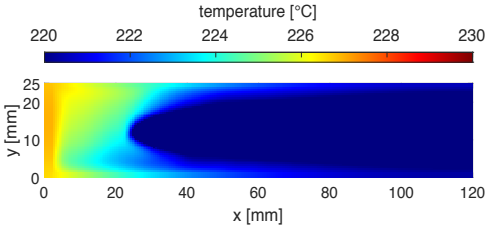
(r) velocity after 22 minutes



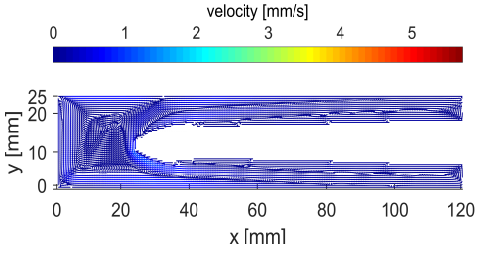
(s) temperature after 45 minutes



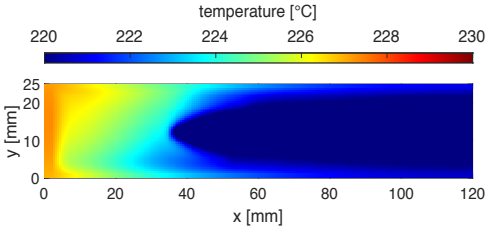
(t) velocity after 45 minutes



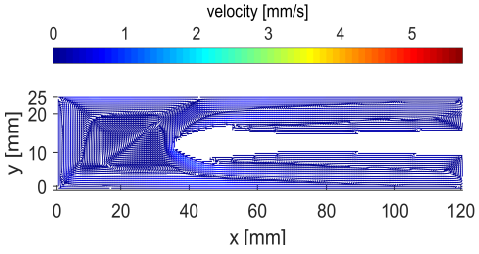
(u) temperature after 67 minutes



(v) velocity after 67 minutes

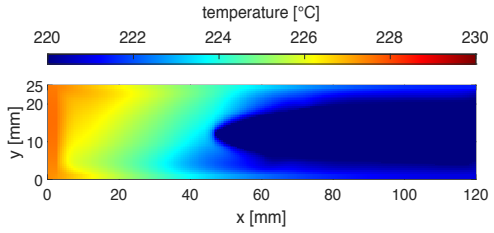


(w) temperature after 90 minutes

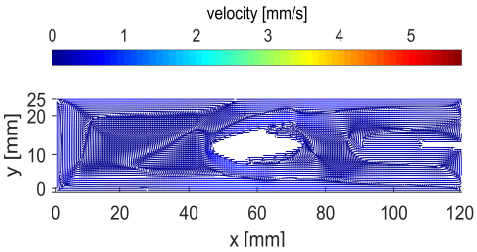


(x) velocity after 90 minutes

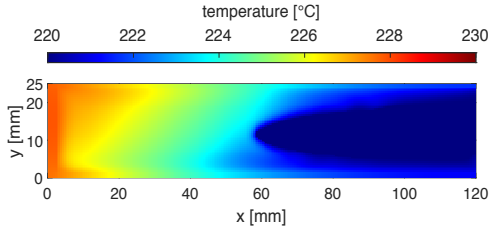
Figure B.5: Temperature and velocity distribution at different simulation times for cavity orientation 180°



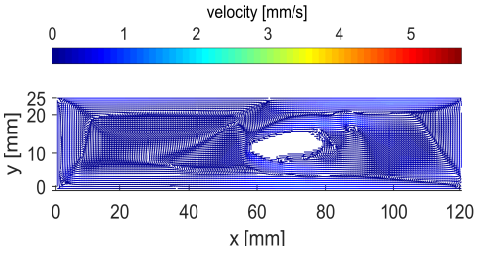
(i) temperature after 112 minutes



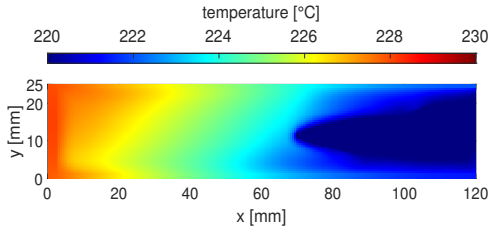
(j) velocity after 112 minutes



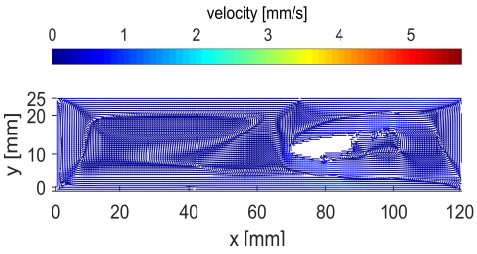
(k) temperature after 135 minutes



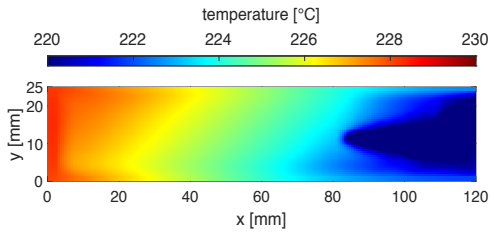
(l) velocity after 135 minutes



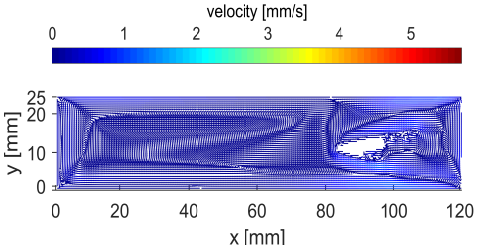
(m) temperature after 157 minutes



(n) velocity after 157 minutes



(o) temperature after 180 minutes



(p) velocity after 180 minutes

Figure B.5: Temperature and velocity distribution at different simulation times for cavity orientation 180°

To increase the efficiency of energy-intensive industrial processes, thermal energy storages can offer new possibilities. A novel approach is investigated in the project HyStEPs. In this concept, containers filled with PCM are placed at the shell surface of a Ruths steam storage, to increase storage efficiency.

In this work, a two-dimensional model using the finite element method is developed to simulate the PCM of the hybrid storage as designed in the HyStEPs project. The apparent heat capacity method is applied in a MATLAB implementation, considering heat transfer by both conduction and natural convection. This successfully validated code can handle any desired layout of materials arranged on a rectangular domain.

Furthermore, a parameter study of different dimensions and orientations of the PCM cavity was conducted. The impact of natural convection was found to lead to significantly varying behaviour of the studied cavities with different orientation during the charging process, while it was found to be negligible during the discharging process.

ISBN 978-3-85448-036-5

www.tuwien.at/academicpress