NUMERICAL ANALYSIS OF THERMAL DISTRIBUTIONS AND LOCALIZED MELTING IN SOLIDS

BY

WILHELM UNKELBACH

NUMERISCHE ANALYSE VON TEMPERATURVERTEILUNGEN UND LOKALEM SCHMELZVERHALTEN IN FESTSTOFFEN

MASTER THESIS

IN

MASCHINENBAU - MECHANICAL AND PROCESS ENGINEERING

UNIVERSITY OF RHODE ISLAND

AND

TECHNISCHE UNIVERSITÄT DARMSTADT

2022

MASTER OF SCIENCE THESIS

OF

WILHELM UNKELBACH

APPROVED:

Thesis Committee:

Major Professor Prof. Dr. David G. Taggart

Apl. Prof. Dr. Sc. Tatiana Gambaryan-Roisman

Prof. Dr.-Ing. Peter Stephan

UNIVERSITY OF RHODE ISLAND

AND

TECHNISCHE UNIVERSITÄT DARMSTADT

2022

ABSTRACT

Analytic solutions for heat transfer problems are limited to simple geometries and boundary conditions. This thesis aims to investigate numerical methods for solving transient heat transfer problems, with particular emphasis on problems involving localized heating that can lead to phase change. Such problems are of increasing importance in modelling additive manufacturing processes. Heat transfer is present in most engineering applications. Complex problems require numerical solutions which involve complex geometries, non-uniform boundary conditions and transient temperature history. For such problems, finite difference and finite element solution techniques have been developed. This thesis introduces the basic numerical models for finite difference and finite element calculations. Then the found calculation problems are employed to solve validation problems and generic heat transfer problems. A comparison is drawn to commercial code, namely Abaqus. Finally, melting and freezing processes are modelled and evaluated.

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Wilhelm Unkelbach, die vorliegende Master-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

English translation for information purposes only:

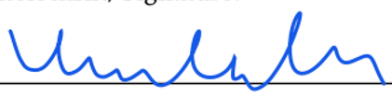
Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Wilhelm Unkelbach, have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

For a thesis of the Department of Architecture, the submitted electronic version corresponds to the presented model and the submitted architectural plans.

Datum / Date: 17.10.2022

Unterschrift/Signature: 

Numerical Analysis of Thermal Distributions and Localized Melting in Solids

Numerische Analyse von Temperaturverteilungen und lokalem Schmelzverhalten in Feststoffen

TECHNISCHE UNIVERSITÄT DARMSTADT

Master-Thesis of Wilhelm Unkelbach (matriculation number: 2819914)
Supervisor: Apl. Prof. Dr. Sc. Tatiana Gambaryan-Roisman
Starting date: 19.4.2022

Analytic solutions for heat transfer problems are limited to simple geometries and boundary conditions. Thermal analysis of complex geometries requires solution by numerical methods. This requires the investigation of numerical methods for solving transient heat transfer problems, with particular emphasis on problems involving localized heating that can lead to phase change. Such problems are of increasing importance in modelling additive manufacturing processes. Recent developments in the field of additive manufacturing, in which filaments are heated in extruders and deposited in controlled patterns or powders are melted by controlled motion of a moving heat sources, lead to complex heat transfer probems. Such problems require numerical solutions which involve complex geometries, non-uniform boundary conditions and transient temperature history. For such problems, finite difference and finite element solution techniques have been developed.

Heat transfer analysis of porous material or metal powders are complicated since the high surface area per volume leads to high heat transfer rates. Initial investigation will consider traditional heat transfer problems using Matlab and Abaqus, a commercial finite element code. Cases to be considered include 1- and 2-dimensional solid body problems with given boundary conditions. The generic problems can also include heat sources and convection. After successfully achieving realistic temperature distributions and validating the solution accuracy, the problems will incorporate either stresses or liquid elements.

task outline:

- Literature research
- Calculation of generic problems using Matlab and Abaqus
- Incorporation of phase change or stresses
- Evaluation of the simulations
- Validation using different code or experiments
- Written documentation

Apl. Prof. Dr. Sc. Tatiana Gambaryan-Roisman

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Motivation

Heat transfer is present in most engineering applications and its investigation ranges from household appliances to combustion engines to large-scale thermal power plants. Recent developments in the field of additive manufacturing, in which filaments are heated in extruders and deposited in controlled patterns or powders are melted by controlled motion of a moving heat sources, lead to complex heat transfer problems. Such problems require numerical solutions which involve complex geometries, non-uniform boundary conditions and transient temperature histories. For such complex problems, numerical finite difference and finite element solution techniques can be applied. Many of these techniques have been incorporated into commercial software packages. Melting represents a phase change which involves absorption of energy, volumetric expansion and change in material properties. Energy absorption affects the temperature distribution and thermal expansion can induce stresses and therefore can cause deformation and possible damage. Melting can be modeled through the use of a potential function [5]. The field variables for modeling melting can also be modeled via subroutines in the commercial finite element code, Abaqus. For example Yaakoubi et al. developed an Abaqus USDFLD user subroutine for modeling melting of iron [6]. Heat transfer analysis of porous materials or metal powders are complicated since the high surface area per volume leads to high heat transfer rates [7]. Clearly, the complexity of these problems requires the development of appropriate numerical modeling techniques.

Before discussing details of this investigation, some fundamentals of heat transfer shall be reviewed. Heat conduction is governed by the general heat conduction equation.

$$\frac{dT}{dt} - \alpha \Delta T = \frac{dQ/dt}{\rho c_p} \tag{1}$$

where T is temperature, t the time, Q the heat generated within the solid, c_p the material's specific heat capacity at constant pressure, ρ its density, and α its thermal diffusivity [8]. The thermal diffusivity equals $\frac{k}{\rho c_p}$ where k is the thermal conductivity. Free and forced convection can be included using Newton's Law of cooling

$$q_h = h\ (T - T_\infty) \tag{2}$$

with the convection coefficient h. The term q_h is added on the right-hand side of Eq. 1. In finite element formulations, solving the heat conduction equation numerically is similar to solving stress-displacement problems [3].

In this thesis, numerical methods for 1-dimensional heat transfer problems will be developed and solved in Matlab, with validation by comparing to analytical solutions and/or solutions obtained using Abaqus, a commercial finite element code. For 2-dimensional problems, solutions will be found using Abaqus. In some cases, the boundary conditions are Dirichlet problems [6], i. e. the surface temperature is explicitly given by a function of time (and location). The generic problems can also include heat sources and convection (Neumann problems). After successfully achieving realistic temperature distributions, the soluction accuracy shall be validated.

In order to compute the numerical solution to stress-strain problems and heat transfer in rigid bodies, the underlying equations shall briefly be introduced. Forces acting on a deformable body can be calculated using a global stiffness matrix which essentially models the body as a multitude of nodes connected by effective elastic springs [3]. The individual element stiffnesses are assembled to define the global stiffness matrix. Eq. 3 shows the relationship between the global force vector $\tilde{F}$,

the global stiffness matrix $\tilde{K}$ and the displacement vector d.

$$\overrightarrow{F} = \tilde{K}\overrightarrow{d}. \tag{3}$$

Heat transfer problems can be modelled very similarly. The global conduction matrix K replaces the global stiffness matrix and displacements are replaced by nodal temperatures, T. The global force vector, F, includes specified heat flux and/or convection at the surface. Therefore, the global equation for heat conduction and convection becomes

$$\overrightarrow{F} = K\overrightarrow{T}. \tag{4}$$

In order to numerically solve Eqs. 3 and 4, displacement or temperature interpolation functions, typically polynomials of linear or quadratic order, are used. For heat transfer problems, the relationship between heat flux and temperature gradients, analogous to strain and displacements relations, are defined. The conduction and convection matrices are calculated for each element. This step is analogous to the forces and displacements of each element that are calculated in case of a stress-strain problem. After formulating these equations for each discretized element, the global equations are assembled and boundary conditions are imposed. Finally, this system of global equations is solved for the nodal temperatures which yields the temperature gradients and therefore the heat flux. This step is analogous to the case of stress-strain problems, where the global equations yield the displacement of each node, which can then be used to calculate the strains and stresses. These procedures are detailed for both stress analysis and heat transfer problems in the text by Logan [3].

CHAPTER 2

Derivation of finite difference calculation schemes

2.1 Finite difference in Cartesian coordinates

The finite difference method is commonly used to solve the heat conduction equation numerically. The governing equation is simplified using a difference quotient instead of the derivative. Starting with the heat conduction equation:

$$\alpha \nabla^2 T = \dot{T}. \tag{5}$$

where α is the thermal diffusivity ($k/(\rho c_p)$), ∇^2 is the Laplacian operator, T is temperature, and $\dot{T}$ is $\frac{dT}{dt}$. In Cartesian coordinates, Eq. 5 becomes

$$\alpha \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right) = \frac{dT}{dt}. \tag{6}$$

In the 1-dimensional case, this simplifies to

$$\alpha \frac{\partial^2 T}{\partial x^2} = \frac{dT}{dt}. \tag{7}$$

The difference quotients can be approximated in order to calculate the partial derivatives numerically as follows

$$\frac{\partial T}{\partial x} \approx \frac{T(x + \Delta x, t) - T(x, t)}{\Delta x} \tag{8}$$

and

$$\frac{dT}{dt} \approx \frac{T(x, t + \Delta t) - T(x, t)}{\Delta t}. \tag{9}$$

In order to approximate the second derivative of Eq. (7) a central differencing scheme is used. The difference quotient can also be obtained by using the same structure as in Eq. (8) for $\partial T/\partial x$. This yields

$$\frac{\partial^2 T}{\partial x^2} \approx \frac{T(x + \Delta x, t) - 2T(x, t) + T(x - \Delta x, t)}{\Delta x^2}. \tag{10}$$

Now the spatial coordinate x shall be denoted x_i and the temporal coordinate t_j respectively. The next spatial step is $x_{i+1} = x + \Delta x$ and the time-step analogously $t_{j+1} = t + \Delta t$. Eq. (7) can be discretized in a more compact form as

$$\frac{T(x_i, t_{j+1}) - T(x_i, t_j)}{\Delta t} = \alpha \left(\frac{T(x_{i+1}, t_j) - 2T(x_i, t_j) + T(x_{i-1}, t_j)}{\Delta x^2} \right) \tag{11}$$

By introducing the variable $X_{i,j}$ for $T(x_i, t_j)$ this can be further condensed

$$\frac{X_{i,j+1} - X_{i,j}}{\Delta t} = \alpha \left(\frac{X_{i+1,j} - 2X_{i,j} + X_{i-1,j}}{\Delta x^2} \right) \tag{12}$$

Solving for the next time step yields

$$X_{i,j+1} = \alpha \Delta t \left(\frac{X_{i+1,j} - 2X_{i,j} + X_{i-1,j}}{\Delta x^2} \right) + X_{i,j} \tag{13}$$

This scheme can be applied iteratively to calculate the temperature for each consecutive time step. A code example for a later discussed problem is attached in Appendix A.

2.2 Finite difference in spherical coordinates

The same procedure as in the previous section can be employed to simplify heat conduction problems in spherical coordinates. This is helpful for problems with spherical symmetry (e.g. a heat source in the center of a solid mass). The heat conduction equation (Eq. 5) can be expressed in spherical coordinates as

$$\frac{\alpha}{r^2} \left(\frac{\partial}{\partial r} \left(r^2 \frac{\partial T}{\partial r} \right) + \frac{1}{sin(\varphi)} \frac{\partial}{\partial \varphi} \left(sin(\varphi) \frac{\partial T}{\partial \varphi} \right) + \frac{1}{sin^2(\varphi)} \frac{\partial^2 T}{\partial \vartheta^2} \right) = \frac{dT}{dt} \tag{14}$$

which, for spherically symmetric problems, reduces to

$$\frac{\alpha}{r^2} \left(\frac{\partial}{\partial r} \left(r^2 \frac{\partial T}{\partial r} \right) \right) = \frac{dT}{dt} \tag{15}$$

using the identities $\frac{\partial}{\partial \varphi} = \frac{\partial}{\partial \vartheta} = 0$. This assumption means that there is no temperature dependence on coordinates φ and ϑ. Using the product rule, Eq. (15) can be discretized as

$$\alpha \left(\frac{2}{r} \frac{\partial T}{\partial r} + \frac{\partial^2 T}{\partial r^2} \right) = \frac{dT}{dt} \tag{16}$$

The difference quotients can be formulated in order to calculate the partial derivatives numerically as follows

$$\frac{\partial T}{\partial r} \approx \frac{T(r+\Delta r,t)-T(r,t)}{\Delta r} \tag{17}$$

and

$$\frac{dT}{dt} \approx \frac{T(r,t+\Delta t)-T(r,t)}{\Delta t} \tag{18}$$

In order to approximate the second derivative of Eq. (16) a central differencing scheme is used. The difference quotient can also be obtained by using the same structure as in Eq. (17) for $\partial T/\partial r$. This yields

$$\frac{\partial^2 T}{\partial r^2} \approx \frac{T(r+\Delta r,t)-2T(r,t)+T(r-\Delta r,t)}{\Delta r^2} \tag{19}$$

Now the spatial coordinate r shall be denoted r_i and the temporal coordinate t_j respectively. The next spatial step is $r_{i+1} = r + \Delta r$ and the time-step analogously $t_{j+1} = t + \Delta t$. Eq. (16) can be discretized in a more compact form as

$$\frac{T(r_i,t_{j+1})-T(r_i,t_j)}{\Delta t} = \alpha\left(\frac{2}{r_i}\frac{T(r_{i+1},t_j)-T(r_i,t_j)}{\Delta r}+\frac{T(r_{i+1},t_j)-2T(r_i,t_j)+T(r_{i-1},t_j)}{\Delta r^2}\right) \tag{20}$$

By introducing the variable $X_{i,j}$ for $T(r_i,t_j)$ this can be further reduced to

$$\frac{X_{i,j+1}-X_{i,j}}{\Delta t} = \alpha\left(\frac{2}{r_i}\frac{X_{i+1,j}-X_{i,j}}{\Delta r}+\frac{X_{i+1,j}-2X_{i,j}+X_{i-1,j}}{\Delta r^2}\right) \tag{21}$$

Solving for the next time step yields

$$X_{i,j+1} = \alpha\Delta t\left(\frac{1}{r_i}\frac{X_{i+1,j}-X_{i,j}}{\Delta r}+\frac{X_{i+1,j}-2X_{i,j}+X_{i-1,j}}{\Delta r^2}\right)+X_{i,j} \tag{22}$$

This is an analogous expression to Eq. 13. The only difference is the $1/r_i$ term which stems from the product rule (Eq. 16). This equation can be used to solve 1-dimensional heat transfer problems with spherical symmetry. A code example for a later discussed problem is attached in Appendix B.

CHAPTER 3

Finite element formulation

3.1 Introduction

The general form of the heat equation can be expressed as

$$\rho c_p \frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T) + Q \tag{23}$$

where T is the temperature, t is the time, ρ is the density, c_p is the specific heat, k is the thermal conductivity and Q is the internal heat generation. In this chapter, 1-dimensional solutions are obtained in Cartesian, cylindrical and spherical coordinates, with spatial variables defined in Figure 1.

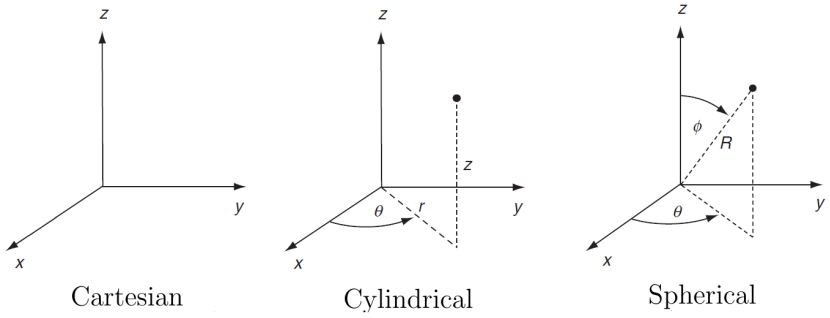


Figure 1: Cartesian, cylindrical and spherical coordinate systems (adapted from [1])

For the case of steady state heat conduction with no internal heat generation and temperature independent thermal conductivity, Eq. 23 reduces to Laplace's equation

$$\nabla^2 T = 0 \tag{24}$$

In Cartesian coordinates, if the spatial variation of temperature is limited to the x-direction, Laplace's equation becomes

$$\nabla^2 T = \frac{d^2 T}{dx^2} = 0 \tag{25}$$

In cylindrical coordinates, if the spatial variation of temperature is limited to the r-direction, Laplace's equation becomes

$$\nabla^2 T = \frac{1}{r}\frac{\partial}{\partial r}\Big(r\frac{dT}{dr}\Big) = 0 \tag{26}$$

In spherical coordinates, if the spatial variation of temperature is limited to the R-direction, Laplace's equation becomes

$$\nabla^2 T = \frac{1}{R^2}\frac{\partial}{\partial R}\Big(R^2\frac{dT}{dR}\Big) = 0 \tag{27}$$

3.2 Derivation of 1-dimensional equations

A brief derivation of application of Galerkin's method for developing 1-dimensional finite element equations is discussed below. Starting with Fourier's Law of Heat Conduction, which is defined as

$$q = -k\frac{dT}{dx} \tag{28}$$

where k is the thermal conductivity, q is the heat flux and T is the temperature, which is assumed to depend only on the spatial direction x, the energy balance on a control volume for steady conduction leads to

$$\frac{d}{dx}\Big(k\frac{dT}{dx}\Big) + Q = 0. \tag{29}$$

In the case of constant thermal conductivity and no heat sources or sinks, Eq. 29 can be further simplified to

$$k\frac{d^2T}{dx^2} = 0. \tag{30}$$

In weighted residual methods (see Logan [3]), $\frac{d^2T}{dx^2}$ is multiplied by a weighting function and integrated over the length of the element. In Galerkin's method, the weighting function is taken to be the finite element temperature interpolation functions, N_i as follows

$$\int_0^L k\Big(\frac{d^2T}{dx^2}\Big)N_i(x)dx = 0 \tag{31}$$

where $i = 1, 2$, $N_1 = 1 - x/L$ and $N_2 = x/L$. This expression can be simplified by using integration by parts as follows

$$\int uv' = uv - \int u'v \tag{32}$$

where $u = N_i$, $u' = du/dx = dN_i/dx$, $v' = dv/dx = kd^2T/dx^2$ and $v = kdT/dx$. With these substitutions, Eq. 31 can be simplified as follows

$$\int_0^L k\Big(\frac{d^2T}{dx^2}\Big)N_i dx = N_i\Big(\frac{dT}{dx}\Big)\Big|_0^L - \int_0^L k\Big(\frac{dT}{dx}\Big)\Big(\frac{dN_i}{dx}\Big)dx = 0. \tag{33}$$

Using

$$T(x) = N_1(x)T_1 + N_2(x)T_2 \tag{34}$$

and

$$\frac{dT}{dx} = \frac{dN_1}{dx}T_1 + \frac{dN_2}{dx}T_2 = \begin{pmatrix}-1/L & 1/L\end{pmatrix}\begin{pmatrix}T_1\\T_2\end{pmatrix} \tag{35}$$

Eq. 33 (for $i = 1$) can be simplified to

$$\int_0^L Ak\frac{dN_1}{dx}\frac{dT}{dx}dx = Ak\int_0^L (-1/2)\begin{pmatrix}-1/L & 1/L\end{pmatrix}\begin{pmatrix}T_1\\T_2\end{pmatrix}dx = \frac{Ak}{L}\begin{pmatrix}1 & -1\end{pmatrix}\begin{pmatrix}T_1\\T_2\end{pmatrix}. \tag{36}$$

Defining $f_{1x} = N_1\Big(\frac{dT}{dx}\Big)\Big|_0^L$ gives the result

$$f_{1x} = \frac{Ak}{L}\begin{pmatrix}1 & -1\end{pmatrix}\begin{pmatrix}T_1\\T_2\end{pmatrix}. \tag{37}$$

A similar expression for f_{2x}, defined as $N_2\Big(\frac{dT}{dx}\Big)\Big|_0^L$, can be obtained, leading to element equations

$$K\ \overrightarrow{T} = \frac{Ak}{L}\begin{pmatrix}1 & -1\\-1 & 1\end{pmatrix}\begin{pmatrix}T_1\\T_2\end{pmatrix} = \begin{pmatrix}f_{1x}\\f_{2x}\end{pmatrix} = \overrightarrow{F} \tag{38}$$

where the 2x2 matrix on the left hand side that multiplies the nodal temperatures is the element conductivity matrix, K. This matrix is analogous to the element stiffness matrix in structural problems.

The global conductivity matrix can be assembled using element conductivity terms a_{ii}, a_{ij}, a_{ji} and a_{jj} for element i which lies between nodes i and j. In the case of a 1-dimensional model, the global conductivity matrix becomes a band matrix with entries on the diagonal of width 2. An example is presented in the next chapter to illustrate the finite element calculation for transient problems and is compared to a finite difference calculation for validation purposes.

In addition to 1-dimensional problems in Cartesian coordinates, it is of interest to develop a finite element formulation for 1-dimensional problems in cylindrical and spherical coordinates. In cylindrical coordinates, when the integrand depends only on the radial coordinate r, integration over the element volume is given by

$$\iiint_V f(r)\, dV = 2\pi h \int_{r_1}^{r_2} \mathrm{r}\, f(r)\, \mathrm{dr} \tag{39}$$

where r_1 and r_2 are the inner and outer radii of the element, respectively, and h is the element's height in z-direction. In spherical coordinates, when the integrand depends only on the radial coordinate R, integration over the element volume is given by

$$\iiint_V f(R)\, \mathrm{dV} = 4\pi \int_{R_1}^{R_2} R^2\, f(R)\, \mathrm{dR} \tag{40}$$

where R_1 and R_2 are the inner and outer radii of the element, respectively.

For axisymmetric problems where the solution depends only on the radial coordinate r, Laplace's equation in cylindrical coordinates reduces Eq. 26. Analogously, for spherically symmetric problems where the solution depends only on the radial coordinate R, Laplace's equation reduces to Eq. 27. Applying Galerkin's method using these forms of Laplace's equations leads to the following element conductivity matrices. For 1-dimensional axisymmetric problems, the element conductivity matrix is given by

$$K = \frac{\pi k (r_1 + r_2) h}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{41}$$

For spherically symmetric problems

$$K = \frac{4\pi k \left(R_1^2 + R_1 R_2 + R_2^2\right)}{3L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{42}$$

Also, it should be noted here that in formulating transient problems, the term on the left hand side of Eq. 23 leads to definition of an element mass matrix. For 1-dimensional transient problems in Cartesian coordinates, the element mass matrix is given by (see Chapter 4 for derivation).

$$M = \frac{\rho c_p A L}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \tag{43}$$

For 1-dimensional axisymmetric problems M, is given by

$$M = 2\pi\rho c_p h \begin{bmatrix} \left(\frac{r_1}{4} + \frac{r_2}{12}\right) & \left(\frac{r_1}{12} + \frac{r_2}{12}\right) \\ \left(\frac{r_1}{2} + \frac{r_2}{12}\right) & \left(\frac{r_1}{12} + \frac{r_2}{4}\right) \end{bmatrix}. \tag{44}$$

For spherically symmetric problems

$$M = 4\pi\rho c_p \begin{bmatrix} \left(\frac{R_1^2}{5} + \frac{R_1 R_2}{10} + \frac{R_2^2}{30}\right) & \left(\frac{\frac{-3R_1^5}{5} + 5R_1^4 R_2 - 5R_1 R_2^4 + \frac{3R_1^5}{5}}{60\ (R_2 - R_1)^3}\right) \\ \left(\frac{\frac{-3R_1^5}{5} + 5R_1^4 R_2 - 5R_1 R_2^4 + \frac{3R_1^5}{5}}{60\ (R_2 - R_1)^3}\right) & \left(\frac{R_1^2}{30} + \frac{R_1 R_2}{10} + \frac{R_2^2}{5}\right) \end{bmatrix}. \tag{45}$$

3.3 Validation problems

To validate expressions derived in Section 3.2, a series of problems are considered where the 1-dimensional finite element solutions are compared to known exact solutions. In some cases, the exact solutions are based on analytic solutions. In other cases, the 1-dimensional solutions are compared to solutions obtained using the commercial general purpose code Abaqus. The 1-dimensional finite element equations are implemented in Matlab scripts given in the Appendices.

To validate the derived equations in Cartesian, cylindrical and spherical coordinates, three type of problems are considered. To validate the conductivity matrices, steady state conduction problems are solved and compared to known solutions. A mesh convergence study was performed for every Abaqus solution that

served as a validation for the used Matlab code. Here, the mesh size was refined until the solution converged. This ensured the mesh is adequately refined.

Figures 2 - 4 show that the finite element results match the known solutions for 1-dimensional Cartesian (Figure 2), cylindrical (Figure 3) and spherical (Figure 4) problems.

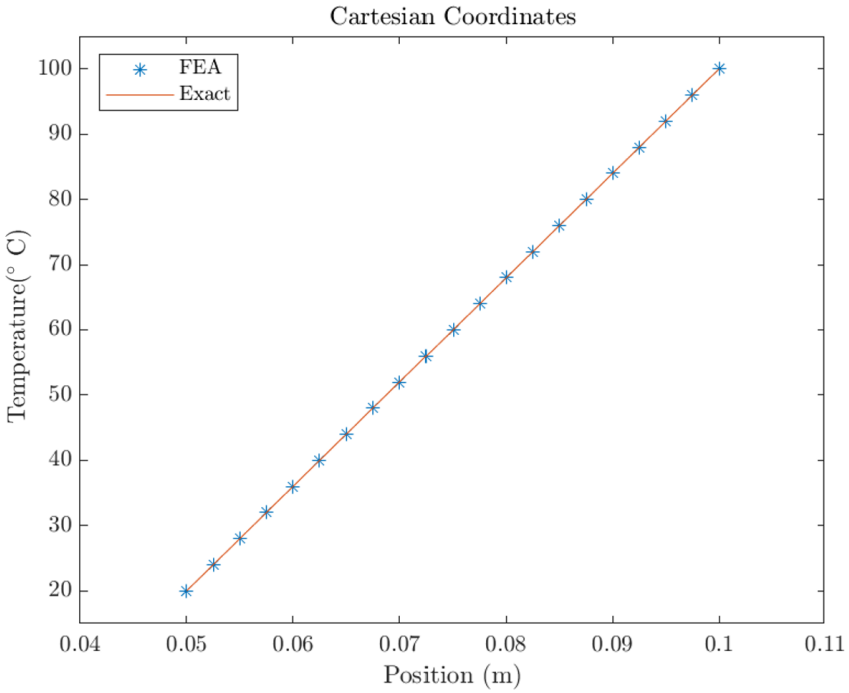


Figure 2: Cartesian Validation Case - Temperature Boundary Conditions

To validate transient problems involving convection boundary conditions, three additional problems are considered. To validate the Cartesian equations, the "Mill's" problem [2] discussed in Chapter 4 is considered. Results for this problem are shown in Figure 5. For the cylindrical formulation, a hollow cylinder is subjected to convection boundary conditions at the inner radius and is insulated at the outer radius. The temperature history at the inner and outer surfaces of the cylinder are shown in Figure 6. Evaluation of the spherically symmetric formula-

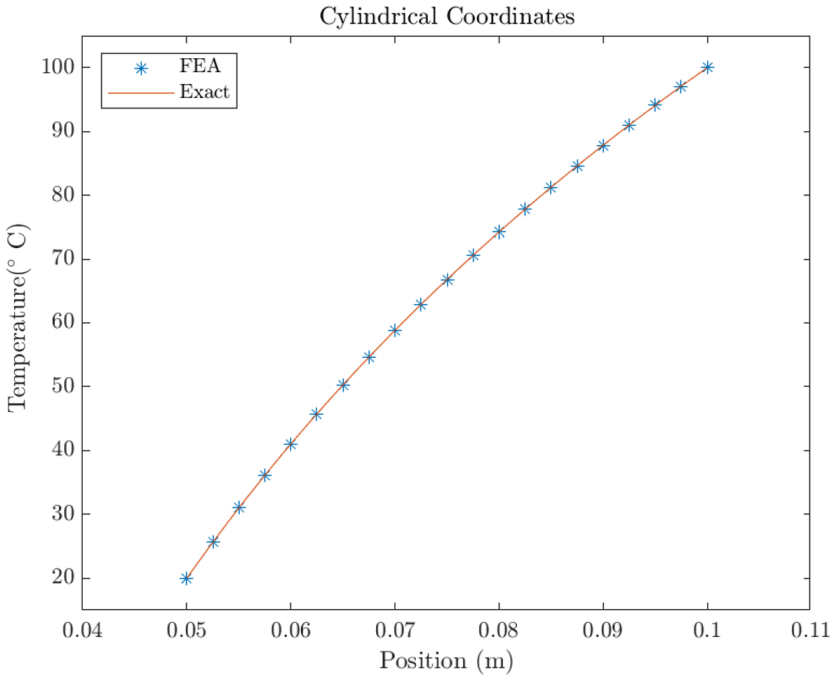


Figure 3: Cylindrical Validation Case - Temperature Boundary Conditions

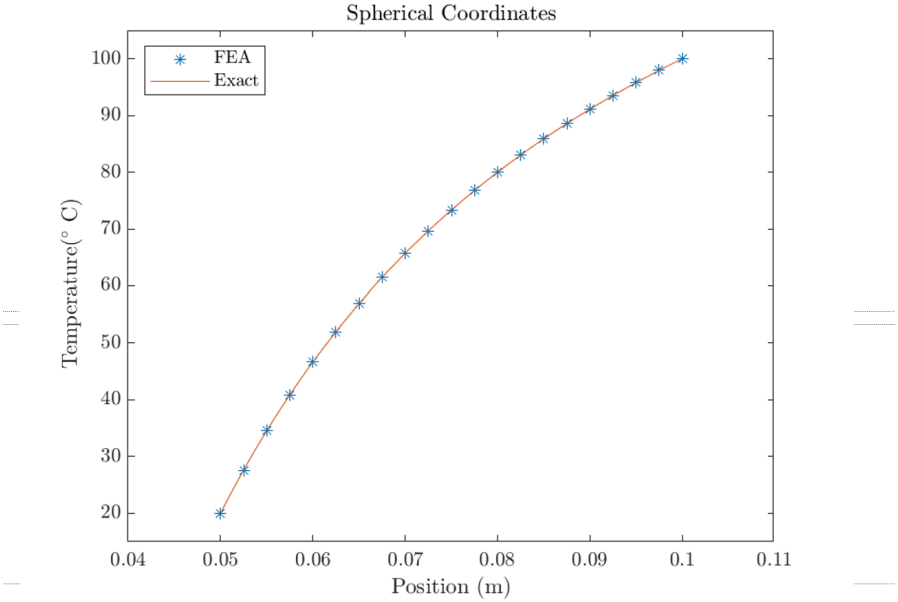


Figure 4: Spherical Validation Case - Temperature Boundary Conditions

tion is obtained using the "boiling egg" problem discussed in Chapter 5. Figure 7 shows that the center of the egg reaches 70 °C in 863 seconds, consistent with the known solution.

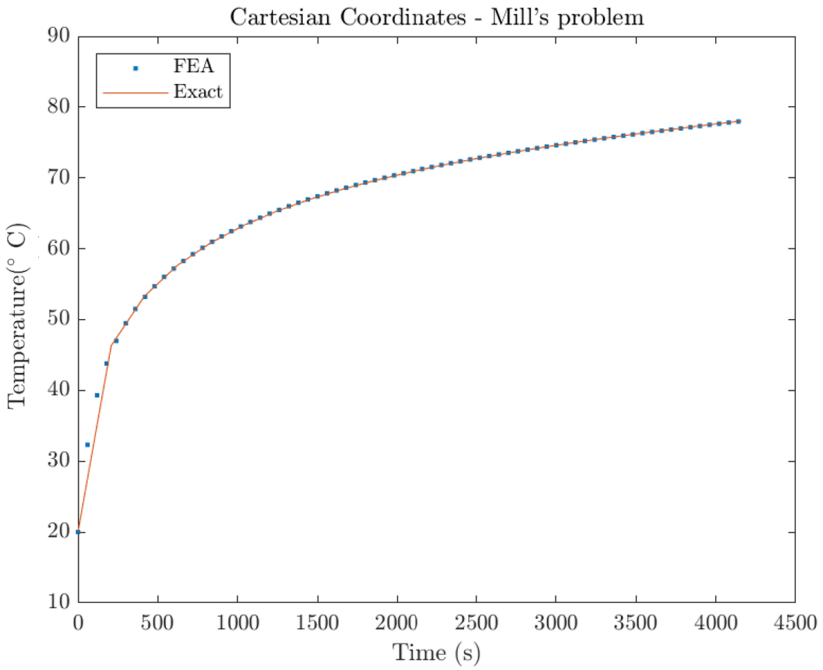


Figure 5: Cartesian Validation Case - Convection Boundary Conditions

To validate transient problems involving specified surface heat flux, three problems are considered. For Cartesian coordinates, the geometry and properties for examining melting of a PLA cylinder (see Chapter 6) are considered here, while ignoring the effects of latent heat of melting, which is considered in Chapter 6. Note that ignoring the effect of latent heat of melting results in unrealistically high temperatures. Despite being unrealistic, this case provides a problem that can be compared to a known solution. The results of this analysis are shown in Figure 8, where a temperature increase of over 3500 °C is observed in 2 seconds, in agreement with the known solution. For the cases of cylindrical and spherical coordinates, cases of a hollow cylinder and a hollow sphere are considered, with

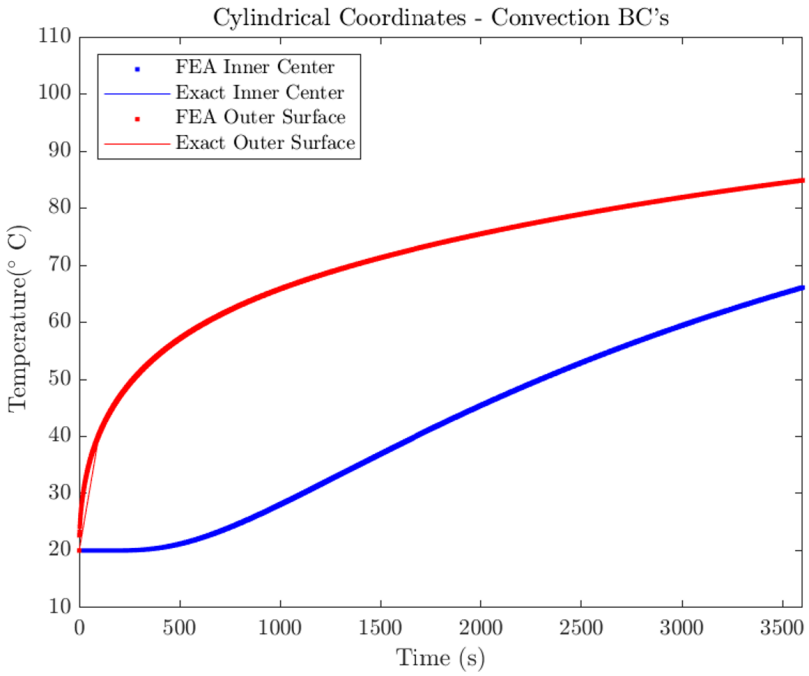


Figure 6: Cylindrical Validation Case - Convection Boundary Conditions

a specified heat flux at the inner radius and insulated boundary conditions at the outer radius. As shown in Figures 9 and 10, the temperature histories at the inner and outer radii match the known solutions.

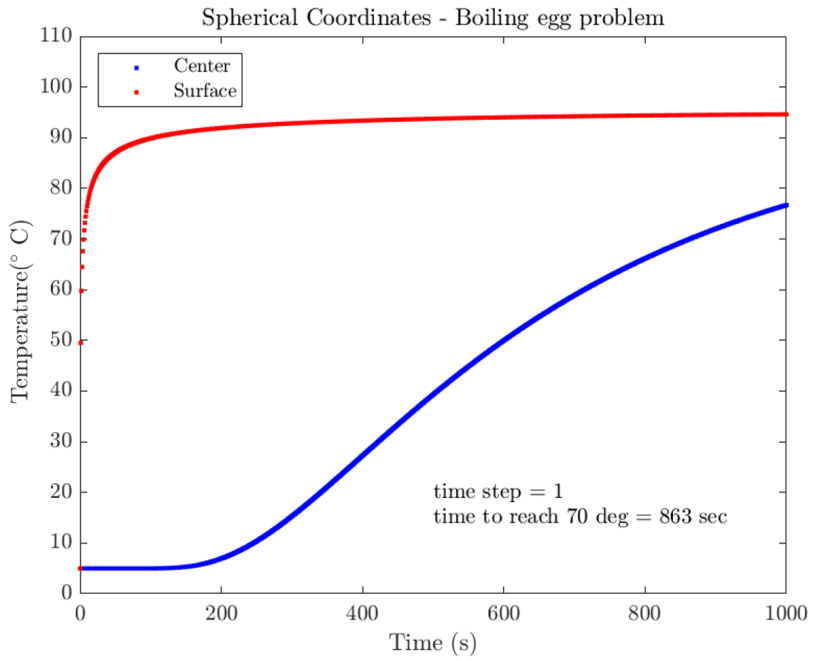


Figure 7: Spherical Validation Case - Convection Boundary Conditions

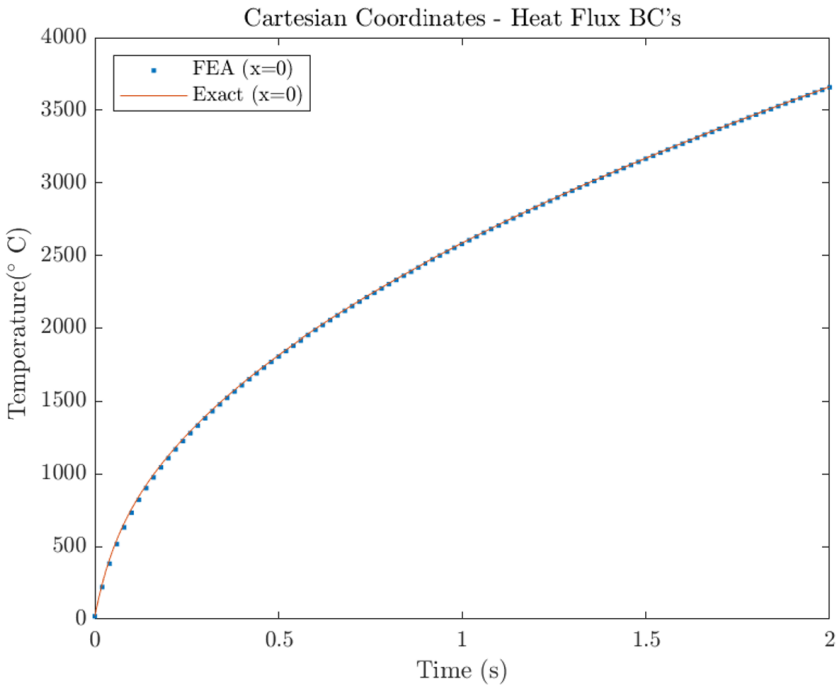


Figure 8: Cartesian Validation Case - Heat Flux Boundary Conditions

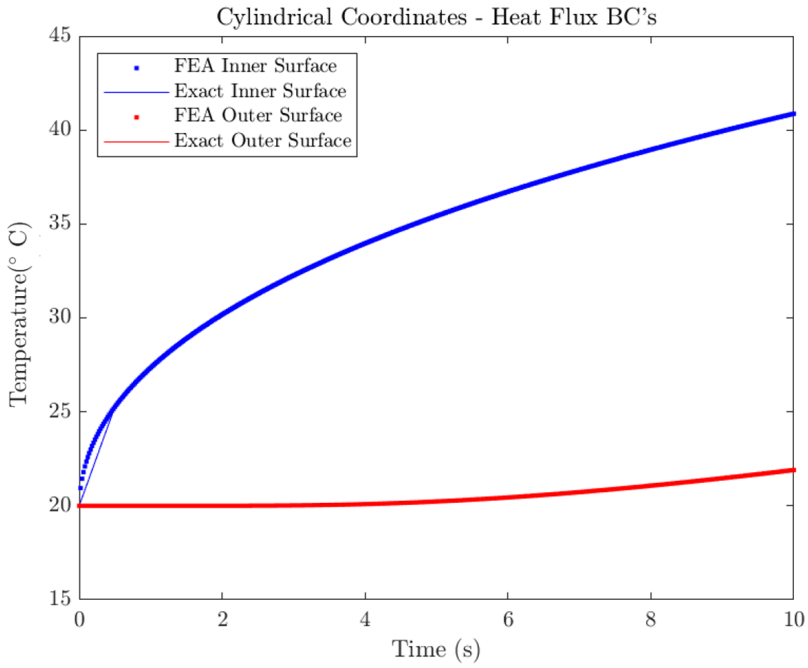


Figure 9: Cylindrical Validation Case - Heat Flux Boundary Conditions

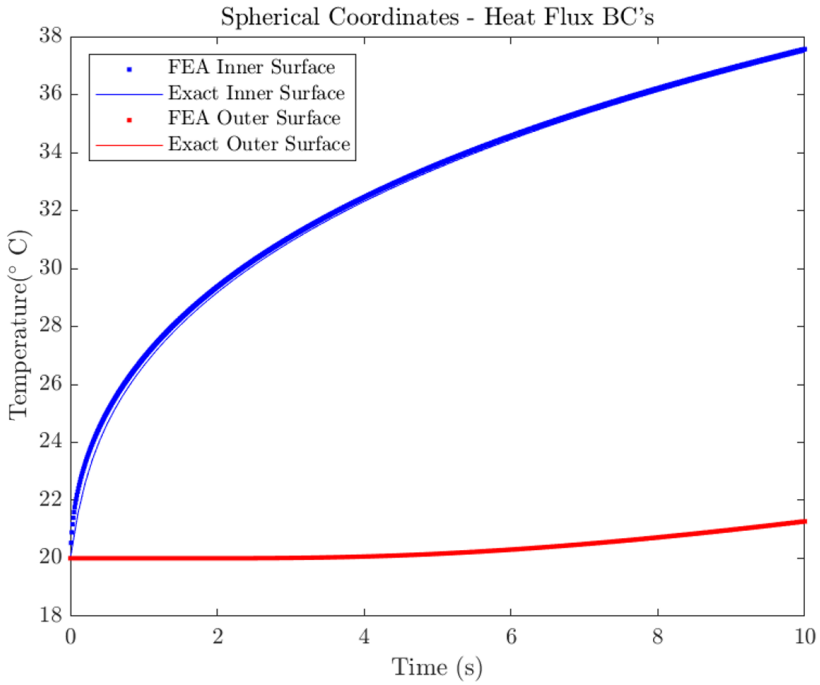


Figure 10: Spherical Validation Case - Heat Flux Boundary Conditions

CHAPTER 4

Convection boundary condition test case

A convective boundary finite difference calculation can be obtained by equating the internal increase in energy with the volumetric conduction as well as the convection across the boundary. The surrounding temperature T_∞ forms a heat sink for the internal temperatures at two consecutive nodes T^i and T^{i+1}. This leads to Eq. 46 [2]:

$$\rho c_p \frac{\Delta x}{2}(T_0^{i+1} - T_0^i) = k\frac{T_1^i - T_0^i}{\Delta x}\Delta t + h(T_\infty^i - T_0^i)\Delta t. \tag{46}$$

It is useful to non-dimensionalize Eq. 46 and solve for T_0^{i+1} similar to the procedure explained in the previous chapter:

$$T_0^{i+1} = 2Fo(T_1^i + BiT_\infty^i) + (1 - 2Fo - 2FoBi)T_0^{i+1} \tag{47}$$

where the Fourier number is defined as $Fo = \alpha\Delta t/\Delta x^2 = k\Delta t/c_p\rho\Delta x^2$ and the Biot number is defined as $Bi = h\Delta x/k$ [9]. This method has been implemented in the Matlab code shown in Appendix D and results are compared to finite element results below.

Alternatively, the finite element method can be adapted to include the temperature change vector $\dot{T}$. The structure of the following equation is similar to Eq. 38. For transient heat transfer problems the discretized energy balance is given by

$$K\overrightarrow{T} + M\overrightarrow{\dot{T}} = \overrightarrow{f_Q} + \overrightarrow{f_q} + \overrightarrow{f_h} \tag{48}$$

with the stiffness matrix K, mass capacitance matrix M, temperature vector T, temperature change vector $\dot{T}$ and force vectors f_Q, f_q, f_h for internal heat sources, heat flux and convection, respectively. Considering the example of a slab of temperature T_0 which is heated by an air stream of temperature T_∞, Eq. 48 simplifies

to

$$K\overrightarrow{T} + M\overrightarrow{\dot{T}} = \overrightarrow{f_h}. \tag{49}$$

A visualization of a slab which is convectively heated is shown in Figure 11. The

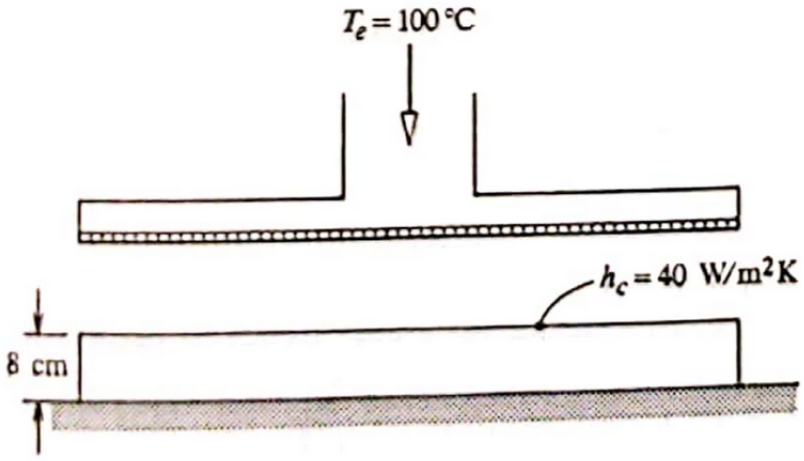


Figure 11: Convectively heated slab (adapted from [2])

stiffness matrix K can be calculated according to the derivation in Chapter 1 (See Eqs. 28 and following), where it is shown

$$K = \int_V B^T DB dV = Ak \int_0^L \begin{pmatrix} -1/L \\ 1/L \end{pmatrix} \begin{pmatrix} -1/L & 1/L \end{pmatrix} dx = \frac{Ak}{L} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}. \tag{50}$$

For the case of a convection boundary condition, the conductivity matrix K is modified due to the nature of Eq. 38. The convection coefficient h multiplies the temperature difference $\Delta T = T_\infty - T_{w(all)}$. In Eq. 38 the force vector F is on the right hand side and does not contain the temperature vector T. Therefore this equation is modified, so $\overrightarrow{T}$ can be included on the left hand side of Eq. 38. The modified conductivity matrix K_h is defined as follows for a convection boundary condition acting on the first node:

$$K_h = K + \tilde{K}_h = \frac{Ak}{L} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} + \begin{pmatrix} hA & 0 \\ 0 & 0 \end{pmatrix}. \tag{51}$$

The mass capacitance matrix M is defined as

$$M = \int_V c_p N^T N dV \tag{52}$$

with the specific heat capacity at constant pressure c_p. For our problem this yields

$$M = \int_V c_p N^T N dV = A c_p \int_0^L \begin{pmatrix} 1 - x/L \\ x/L \end{pmatrix} \begin{pmatrix} 1 - x/L & x/L \end{pmatrix} dx = \frac{c_p A L}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}. \tag{53}$$

The force vector $\overrightarrow{f_h}$ is applied on the outside surface S_1. The right hand side of Eq. 49 therefore yields

$$\overrightarrow{f_h} = \int_{S_1} N^T h T_\infty dS = \frac{A h T_\infty L}{2} \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{54}$$

Following the procedure outlined by Logan [3], the iteration for a future time step of the temperature vector $\overrightarrow{T}_{i+1}$ can be obtained implicitly according to

$$\left(\frac{1}{\Delta t} M + K\right) \overrightarrow{T}_{i+1} = \left(\frac{1}{\Delta t} M - (1 - \beta) K\right) \overrightarrow{T}_i + (1 - \beta) \overrightarrow{F}_i + \beta \overrightarrow{F}_{i+1} \tag{55}$$

using a weighting factor β. For $\beta = 0.5$ this is called the Crank-Nicholson procedure [10]. Using $\beta = 0$ makes the calculation explicit, which improves calculation time but can cause unstable solutions which may diverge. The implicit calculation is guaranteed to always converge.

Results obtained using the finite difference and finite element codes given in Appendices D and E were compared to the commercial finite element code Abaqus. The case considered was defined by Mills [2], with the following problem parameters: $T_0 = 20°\text{C}$, $T_\infty = 100°\text{C}$, $h = 40$ W/m^2 K, $\rho = 2600$ kg/m^3, $c_p = 800$ J/kg K, $k = 1$ W/ K, $N = 5$ nodes and $\Delta t = 60$ s. All three models used the same spatial discretization, with N=41 nodes or 40 elements. An exact solution also exists for this problem and can be found in [2]. The Abaqus simulation was done using the mesh depicted in Figure 12. The final temperature distribution computed by Abaqus after 60 minutes is depicted in Figure 13.

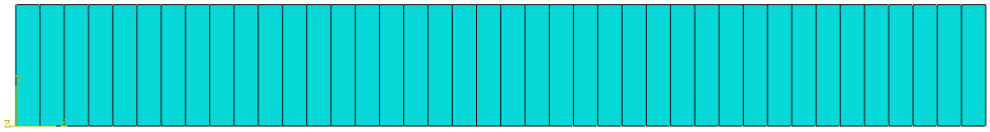


Figure 12: Mesh for Abaqus simulation

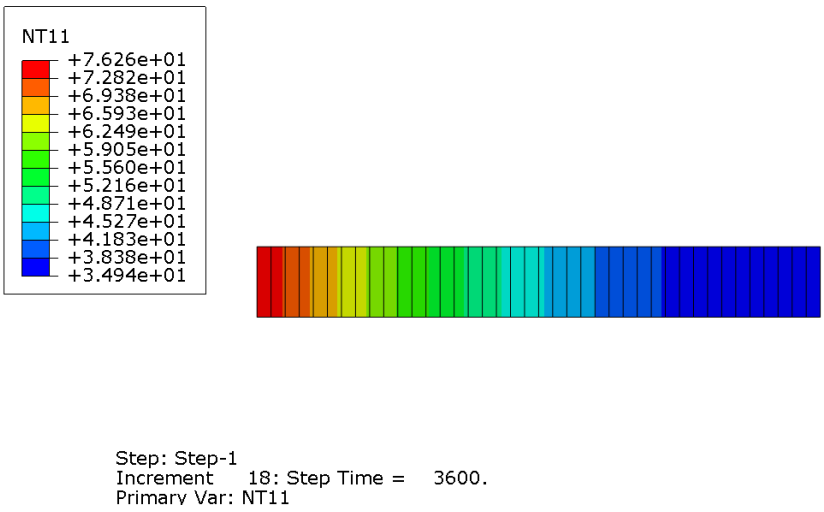


Figure 13: Temperature distribution at $t = 3600$ s $= 60$ min from Abaqus simulation

Comparison of the Abaqus simulation and the finite difference code is shown in Figure 14. The Abaqus temperature distribution in Figure 13 is that same as that plotted in Figure 14. The temperature near the boundary is expectedly high and slowly decreases over the length of the body. As conduction is the only mode of heat transfer within the solid, even after one hour the temperature is only 35 °C at the end of the heated body. The curve in Figure 15 shows the temperature history at the surface exposed to convective heating. As time increases, this surface temperature increases more slowly as it approaches the convective fluid temperature, $T_\infty = 100°$C. All calculations agree very well. The comparison between the

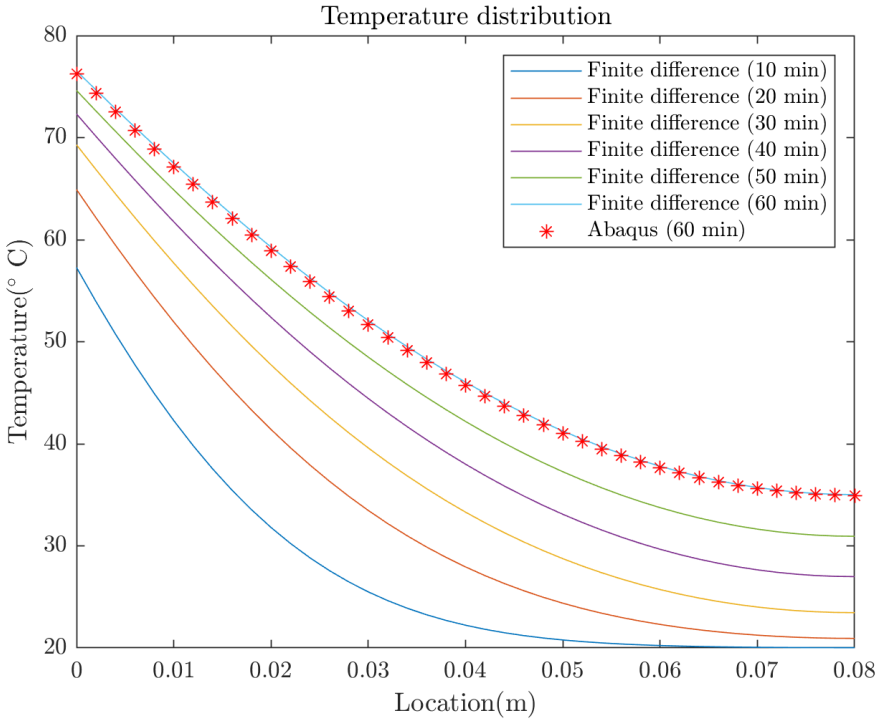


Figure 14: Temperature distribution in Finite Difference calculation and Abaqus simulation

Abaqus simulation and the 1-dimensional finite element code is shown in Figure 15. Here the finite element solution matches the exact solution. In conclusion, for this case of convection boundary conditions, both the 1-dimensional finite difference and finite element methods agree with the Abaqus analysis and the exact solution provided by Mills [2].

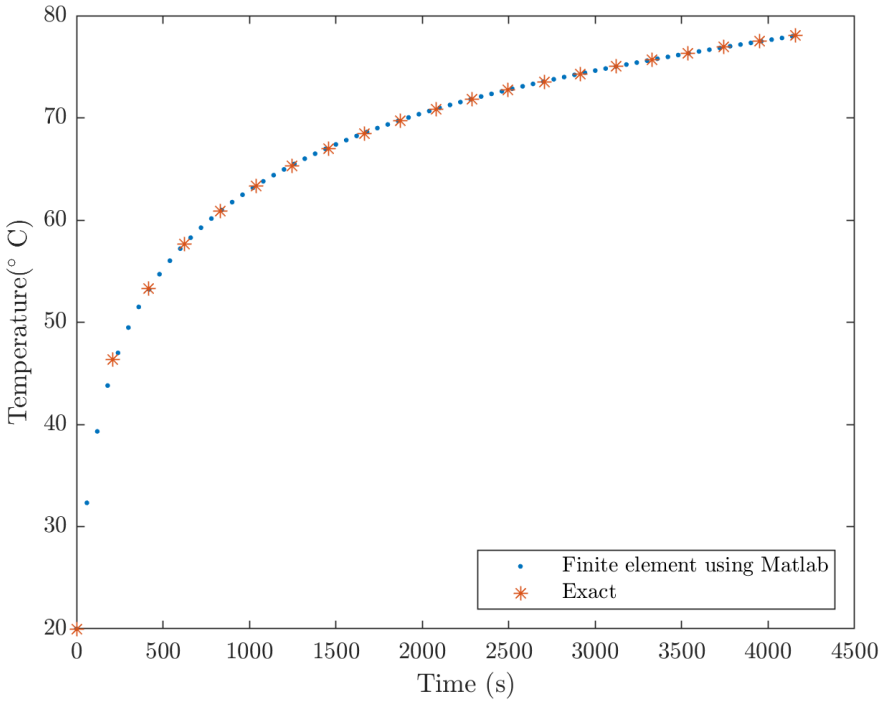


Figure 15: Trend for surface temperature in Finite Element Analysis and exact solution

CHAPTER 5

Solution of various heat transfer problems

5.1 Heated spherical body

The finite difference code for spherically symmetric problems which has been developed in Chapter 2 shall now be employed to calculate the time a sphere takes to heat up when exposed to a higher temperature environment. This problem was solved using a Matlab routine based on Eq. 22, an Abaqus simulation which employs finite element calculations as described in Chapter 2 and compared to the COMSOL solution which is available online [11]. The code used for the Matlab simulation is attached in Appendix B. The radius of the sphere is $r = 2.5$ cm and the initial temperature $T_0 = 5°$C. The calculation was done using both a convection boundary condition and a temperature boundary condition. The wall temperature was $T_w = 95°$ C. For the convection boundary condition a convective film coefficient of $h = 1200$ W/m^2 K was used.

The other constants are density, $\rho = 993.05$ kg/m^3, and specific heat capacity at constant pressure, $c_p = 4178.5$ J/kg K and thermal conductivity $k = 0.627$ W/m K. A transient heat transfer analysis with Abaqus used a time step size of Δt= 1 s, while the Matlab code used Δt= 0.05 s. Both calculations and the COMSOL solution agree as can be seen in Figure 16. Both simulations calculate the time until the outer boundary of the sphere reaches $T_w = 70°$C. COMSOL calculates $t_{final} = 860$ s while the finite difference code determined the time required to be $t_{final} = 875.05$ s.

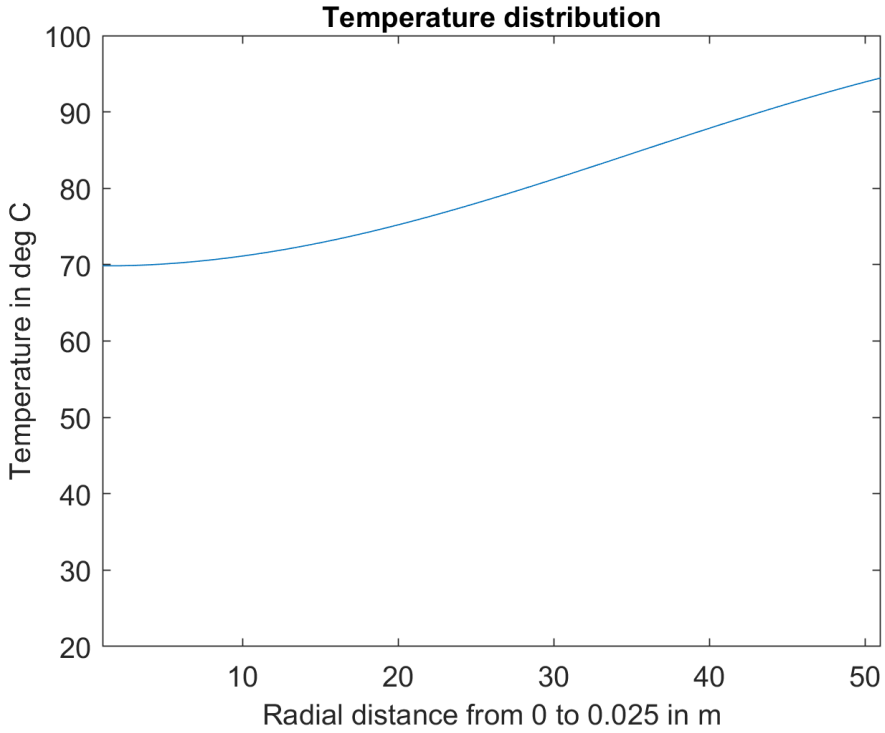


Figure 16: Temperature distribution in finite difference analysis at $t_{final} = 875$ s

5.2 Cooling of insulated rod

The next problem is the steady-state finite element analysis of an insulated rod which has the material properties as delineated in Table 1. The code can be seen in Appendix F.1.

Table 1: Material properties and geometry of insulated rod

T_0	100 °C
T_∞	10 °C
h	10 W/(m^2 K)
k	0.627 W/(m K)
Δx	5/6 m
L	10/3 m

The steady-state solution of this problem leads to a temperature profile of $T_1 = 100\ ^\circ C$ as defined by boundary condition, $T_2 = 85.93\ ^\circ C$, $T_3 = 71.87\ ^\circ C$,

$T_4 = 57.81\ °C$ and $T_5 = 43.75\ °C$ for a spacing of $\Delta x = 5/6\ m$.

5.3 Cooling of rod with free convection

A similar problem to the before discussed one is the cooling of a rod which is now considered with free convection on its perimeter. The material properties are displayed in Table 2. The Matlab code for this problem is shown in Appendix F.2.

Table 2: Material properties and geometry of rod

T_0	200 °C
T_∞	20 °C
h	1 W/(m^{2}K)
k	3 W/(m K)
Δz	3 m
L	9 m
radius, r	2 m
area, A	12.56 m^2
perimeter, P	12.56 m^2

The free convection now results in the term $f_i = h\ T_\infty\ PL/2$ for both components of the force vector. This leads to a final temperature distribution of $T_1 = 200$ °C at the boundary and $T_2 = 42.86$ °C, $T_3 = 22.92$ °C and $T_4 = 20.49$ °C at the nodes.

5.4 Transient insulated fin

Now a non-steady state problem shall be discussed. Similarly to Chapter 3 Eq. 55 is used to calculate the temperature increase over a rod which is heated on one end. A model of the rod with two elements is shown in Figure 17. The material and geometric parameters for this problem are given in Table 3. The Matlab code to solve this problem is attached in Appendix F.3.

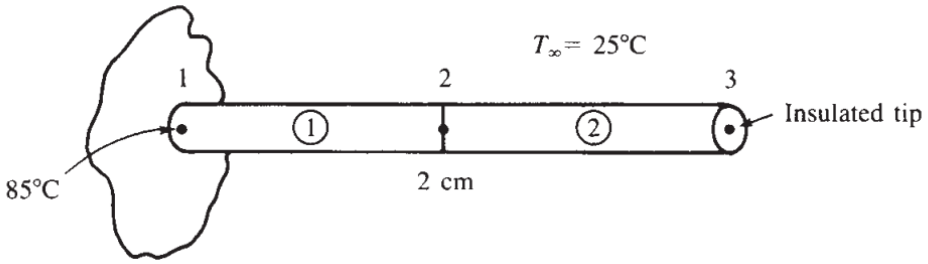


Figure 17: Transient fin with convection (adapted from [3])

Table 3: Material properties and geometry of fin

T_0	25 $°C$
T_∞	25 $°C$
T_1	85 °C
h	150 $W/(m^2\ K)$
k	400 $W/(m\ K)$
Δx	0.01 m
L	0.02 m
radius, r	2 mm
area, A	12.56 mm^2
perimeter, P	12.56 mm

The calculated temperature distribution can be seen in Fig. 18. A slow increase from 25 $°C$ to 70 $°C$ can be observed. As expected, the further node heats up more slowly. A saturation effect similar to the problem discussed in Chapter 4 can be observed. In this case, both nodes tend towards the boundary temperature T_1.

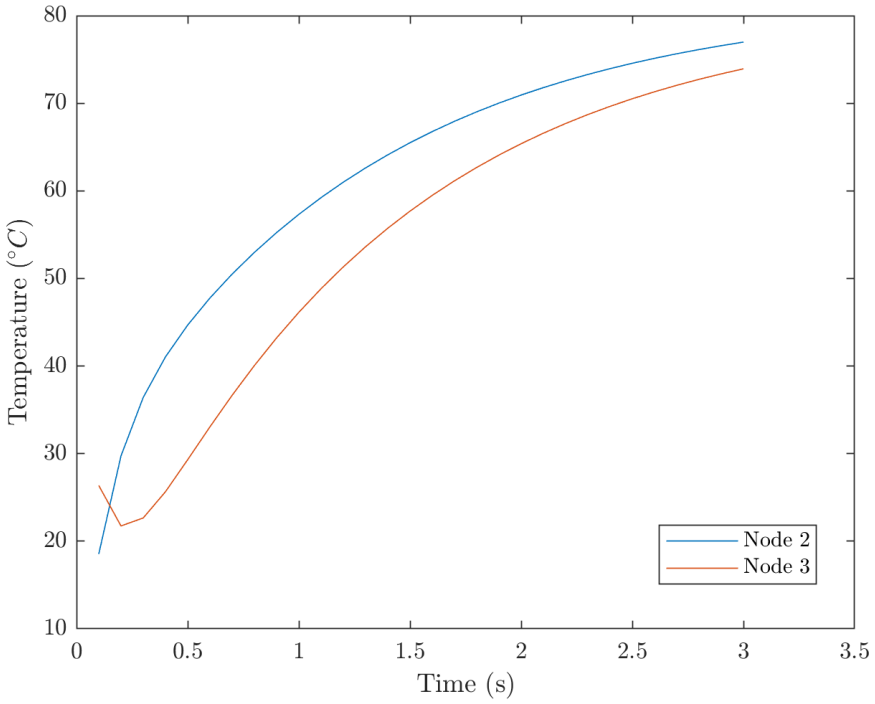


Figure 18: Temperature plot for transient fin for nodes 2 and 3

CHAPTER 6

Heat transfer problems involving phase change

6.1 Freezing problem

This problem was simulated to reproduce the findings of [11]. The problem was considered by performing an Abaqus simulation to verify the literature. A liquid probe is cooled down which leads to freezing from the top of the probe downwards. Points A and B are within the probe and represent different locations. Point A has coordinates $(1, -7)$ and B $(2, -7)$. The freezing boundary condition applies on every edge of a 2-dimensional square with dimensions 8x8. The simulation uses non-dimensionalized units and uses $\rho = c_p = 1$, $k = 1.08$ and a latent heat of $h_{fs} = 70.26$. The location of points A and B as well as the boundary conditions is shown in Figure 19.

The decline in temperature is depicted in Figure 20 and shows good agreement between the Abaqus simulations (for coarse and fine meshes) and the literature solution by [11]. Additionally, Figures 21 to Figures 29 show the growth of the freeze front as well as the temperature distributions at time instants $t_i = \{0, 1, 2, 3, 4, 5\}$. Abaqus used automatic time stepping which determines the time step size according to the residual of a previous iteration. This leads to irregularly spaced time increments which is also depicted in the figures below. The temperature distribution for $t = 0$ s is depicted in Figure 24.

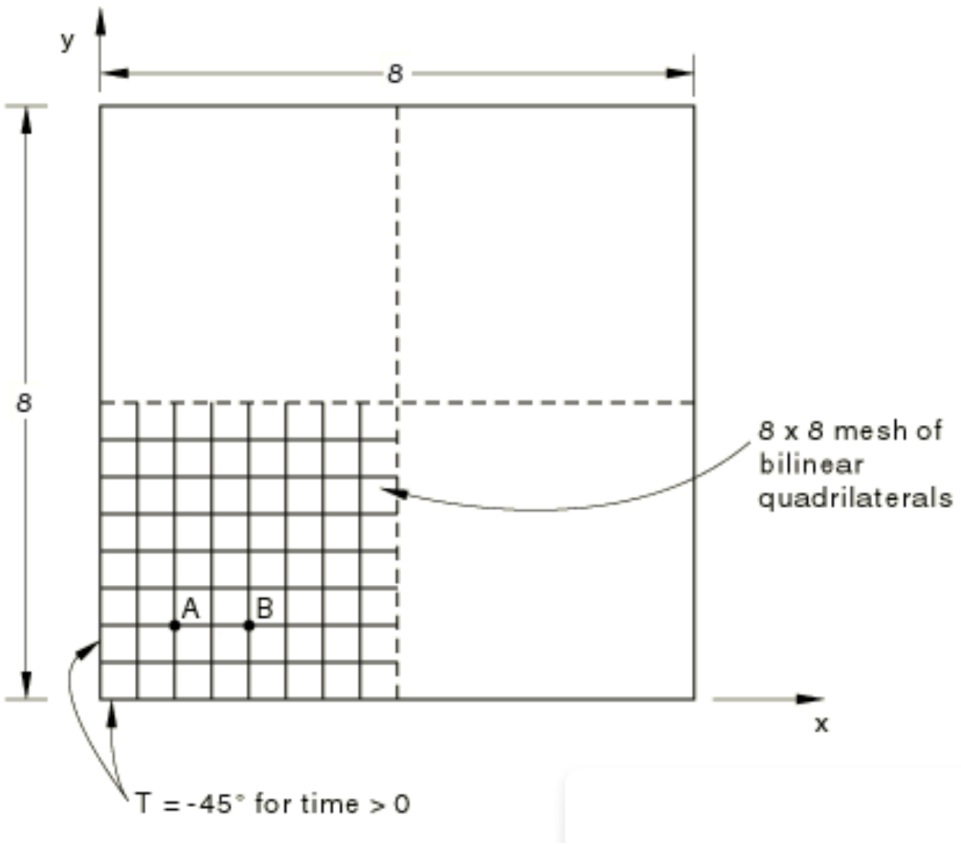


Figure 19: Square plate freezing with points A and B (adapted from [4])

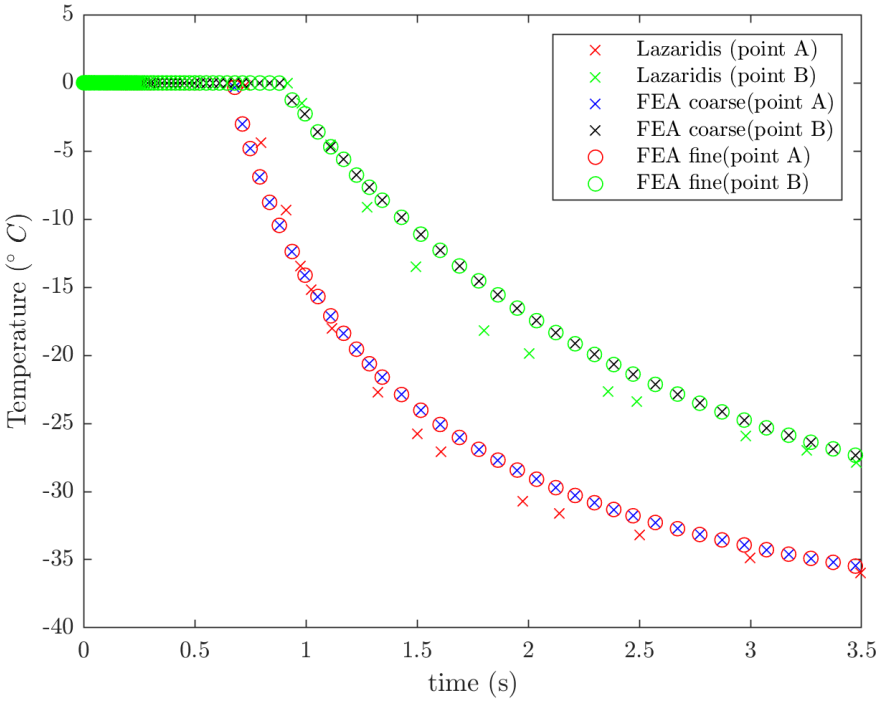


Figure 20: Temperature plot for freezing problem at points A and B

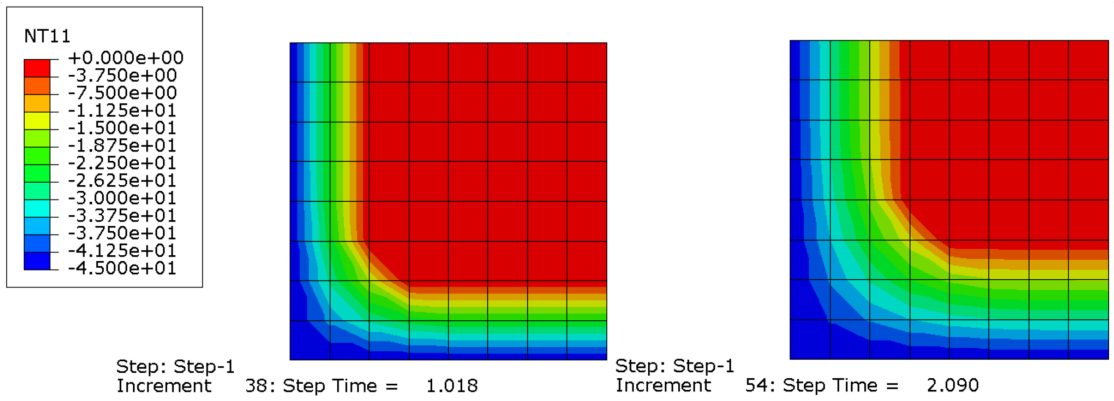


Figure 21: Temperature distribution of freezing problem at $t = 1\ s,\ t = 2\ s$

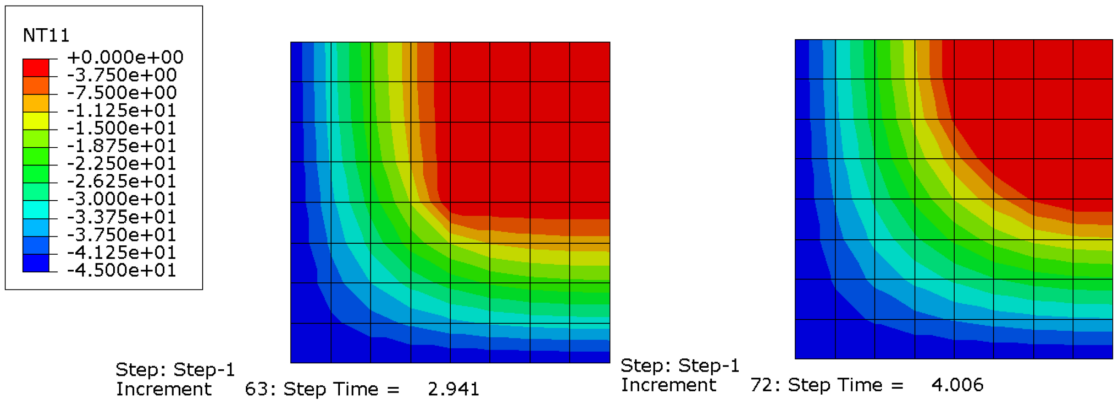


Figure 22: Temperature distribution of freezing problem at $t = 3\ s,\ t = 4\ s$

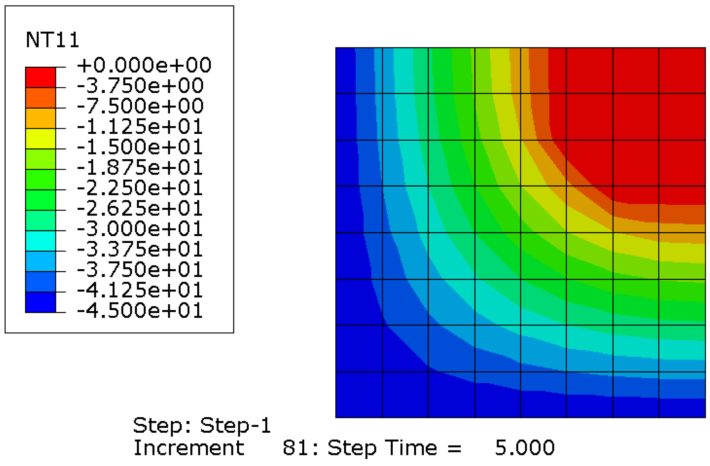


Figure 23: Temperature distribution of freezing problem at $t = 5\ s$

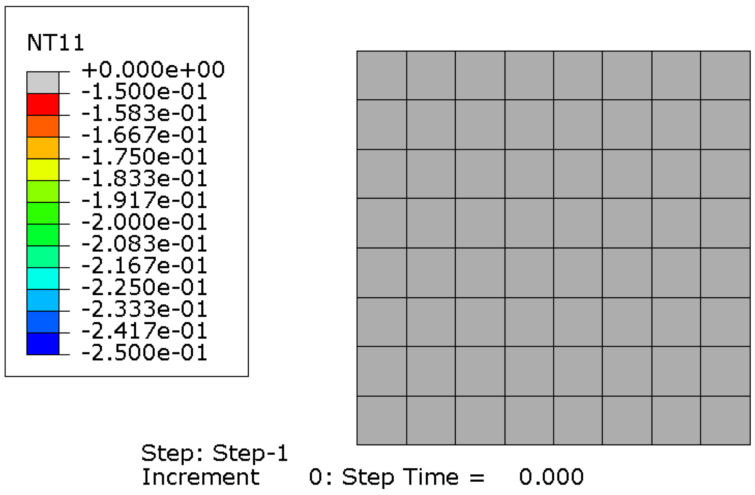


Figure 24: Temperature distibution at $t = 0$ s

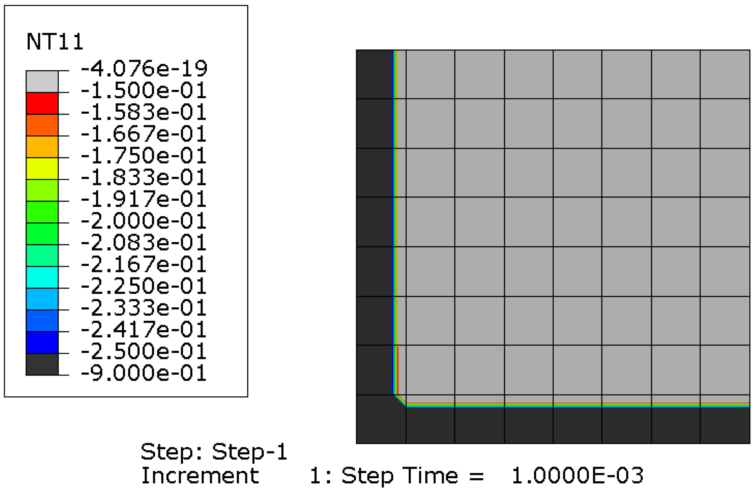


Figure 25: Growth of freeze front (grey to black) at t_1

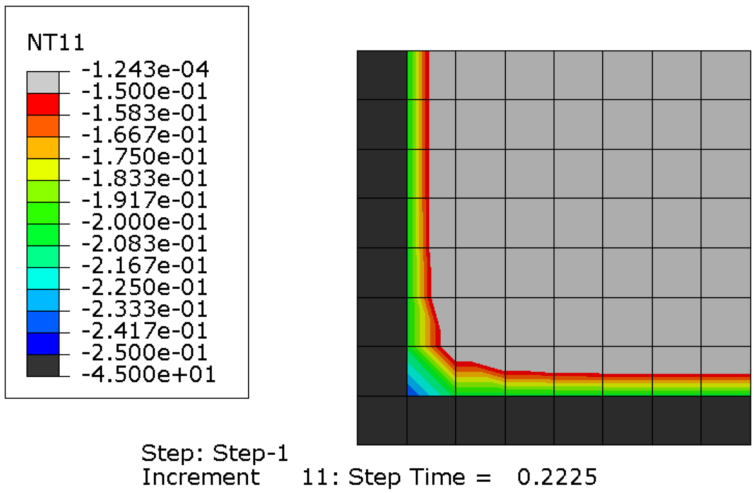


Figure 26: Growth of freeze front (grey to black) at t_2

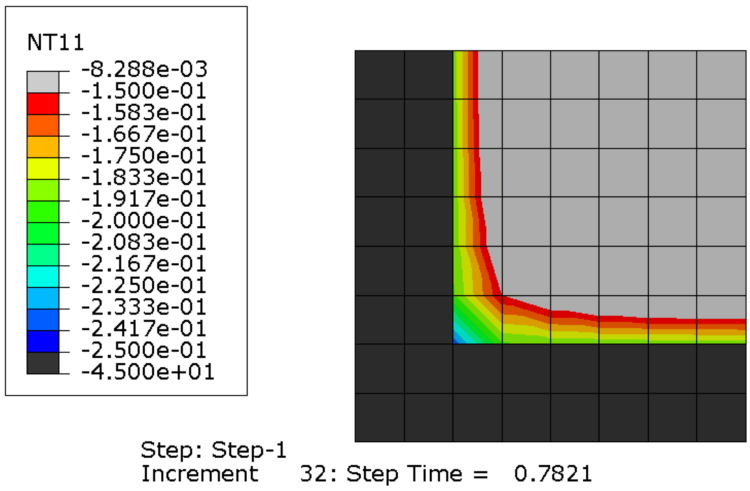


Figure 27: Growth of freeze front (grey to black) at t_3

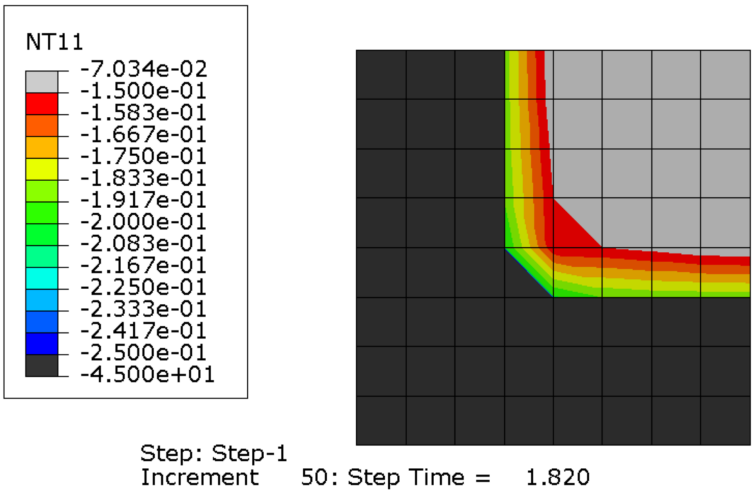


Figure 28: Growth of freeze front (grey to black) at t_4

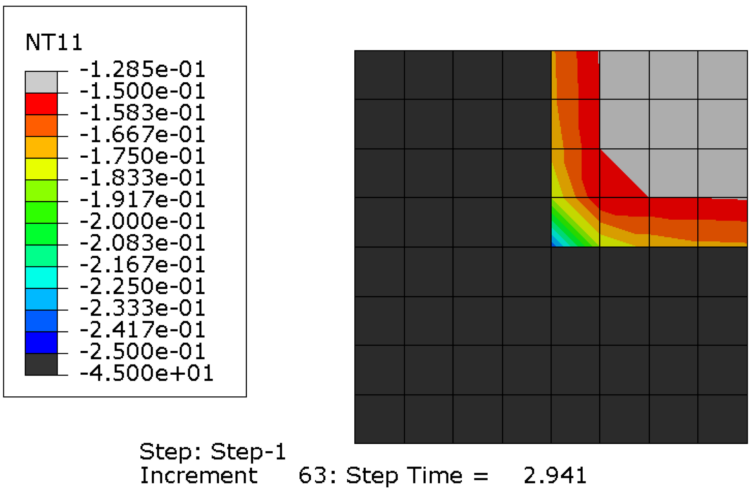


Figure 29: Growth of freeze front (grey to black) at t_5

6.2 1-dimensional phase change with latent heat in force vector

In one approach to modeling phase change, an altered force vector associated with heat released or absorbed can be included in finite element simulations. The liquidus and solidus temperatures define the temperature range over which heat is absorbed or released during the phase change. The solidus temperature is the highest temperature where the material is completely solid and the liquidus temperature is the lowest temperature where the material is completely liquid ($T_S > T_L$). To model phase change for 1-dimensional elements, force vectors can be defined which depend on the temperature distribution within each element.

Within each element, the assumed linear interpolation for the temperature distribution between two nodes with temperatures T_i and T_{i+1} can be expressed as

$$T(x) = \begin{pmatrix} 1 - \frac{x}{L} & \frac{x}{L} \end{pmatrix} \begin{pmatrix} T_i \\ T_{i+1} \end{pmatrix} \tag{56}$$

where x is the position ($0 \leq x \leq L$) and L is the element length. The force vector associated with internal heat generation due to phase change is found by integrating over the values of x for which the temperature is between the solidus and liquidus temperatures:

$$\{f_Q\} = \begin{Bmatrix} f_{Q1} \\ f_{Q2} \end{Bmatrix} = \begin{Bmatrix} h_{\text{fs}} A \displaystyle\int_{x_{\text{lower}}}^{x_{\text{upper}}} \left(1 - \frac{x}{L}\right) \ \text{dx} \\ h_{\text{fs}} A \displaystyle\int_{x_{\text{lower}}}^{x_{\text{upper}}} \frac{x}{L} \ \text{dx} \end{Bmatrix} \tag{57}$$

where h_{fs} is the latent heat of fusion, A is the element cross-sectional area, and x_{lower} and x_{upper} are the limits of integration. The limits of integration correspond to the points where the solidus and liquidus temperatures intersect the interpolated temperature distributions within the element. For each element, depending on the relative values of the nodal temperatures T_1 and T_2, and the solidus and liquidus temperatures, T_S and T_L, 12 possible cases can be identified. These cases are summarized in Table 4 and are illustrated in Figures 30 and 31. The dashed blue line represents T_S and the orange one T_L respectively.

Table 4: Types of element temperature distributions

Type	T_1 vs. T_2	T_1	T_2	x_{lower}	x_{upper}
1	$T_1 > T_2$	$T_1 \leq T_S$	$T_2 \leq T_S$	-	-
2	$T_1 > T_2$	$T_1 > T_S$, $T_1 \leq T_L$	$T_2 \leq T_S$	0	$\frac{(T_1-T_S)L}{(T_1-T_2)}$
3	$T_1 > T_2$	$T_1 > T_S$, $T_1 \leq T_L$	$T_2 > T_S$	0	L
4	$T_1 > T_2$	$T_1 > T_L$	$T_2 < T_S$	$\frac{(T_1-T_L)L}{(T_1-T_2)}$	$\frac{(T_1-T_S)L}{(T_1-T_2)}$
5	$T_1 > T_2$	$T_1 > T_L$	$T_2 \leq T_L$	$\frac{(T_1-T_L)L}{(T_1-T_2)}$	L
6	$T_1 > T_2$	$T_1 > T_L$	$T_2 > T_L$	-	-
7	$T_2 > T_1$	$T_1 < T_S$, $T_1 \leq T_L$	$T_2 \leq T_S$	-	-
8	$T_2 > T_1$	$T_1 \leq T_S$	$T_2 > T_S$, $T_2 \leq T_L$	$\frac{(T_S-T_1)L}{(T_2-T_1)}$	L
9	$T_2 > T_1$	$T_1 \leq T_S$	$T_2 > T_S$, $T_2 \leq T_L$	0	L
10	$T_2 > T_1$	$T_1 \leq T_S$	$T_2 > T_L$	$\frac{(T_S-T_1)L}{(T_2-T_1)}$	$\frac{(T_L-T_1)L}{(T_2-T_1)}$
11	$T_2 > T_1$	$T_1 > T_S$, $T_1 \leq T_L$	$T_2 > T_L$	0	$\frac{(T_L-T_1)L}{(T_2-T_1)}$
12	$T_2 > T_1$	$T_1 > T_L$	$T_2 > T_L$	-	-

For elements where the temperature profile is characterized by types 1, 6, 7 and 12, the element temperatures are either below the solidus temperature or above the liquidus temperature and hence no internal heat is absorbed or released. For the other temperature distribution types, f_Q can be evaluated analytically to compute nodal force values for each element. The resulting nodal force vectors for each case are given in Table 5.

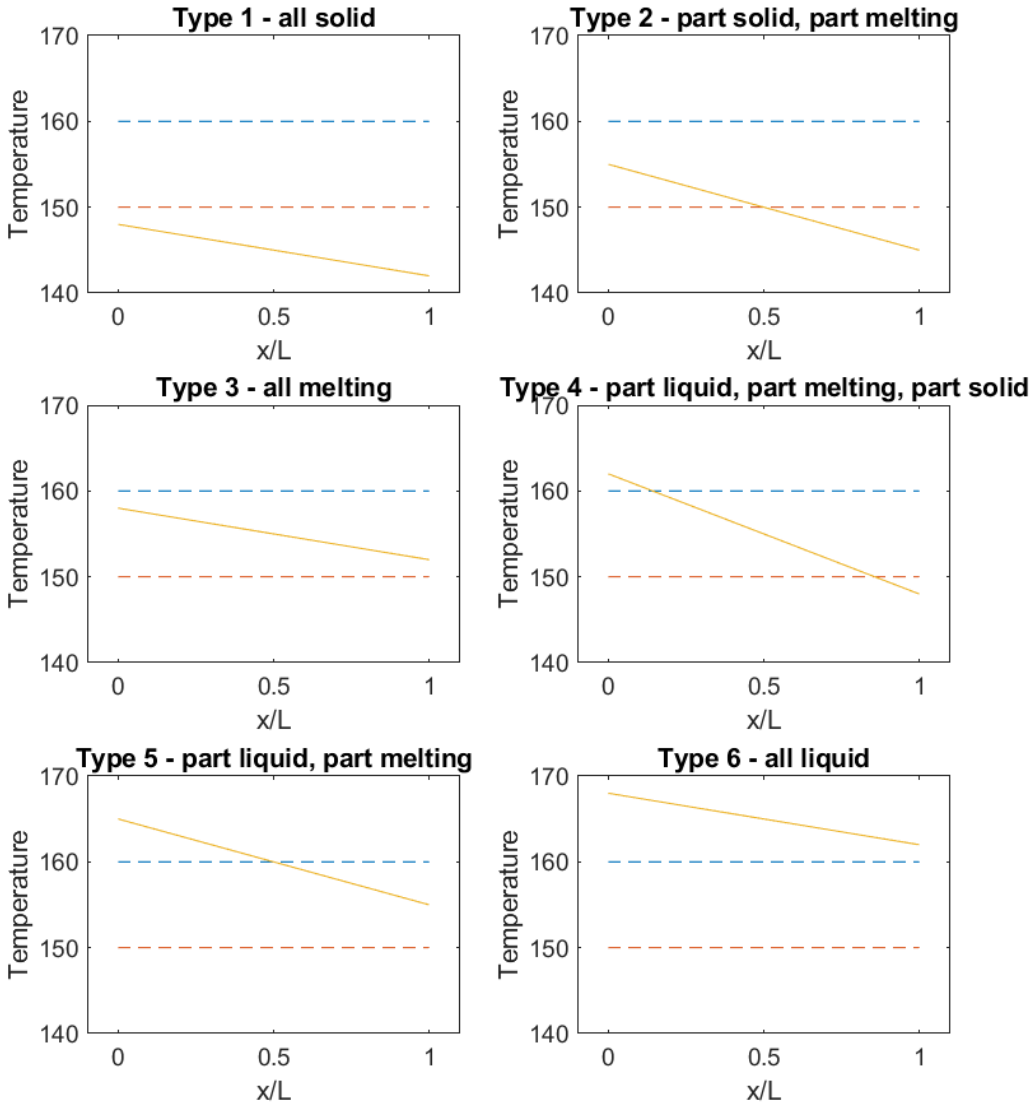


Figure 30: Element temperature distribution types 1-6

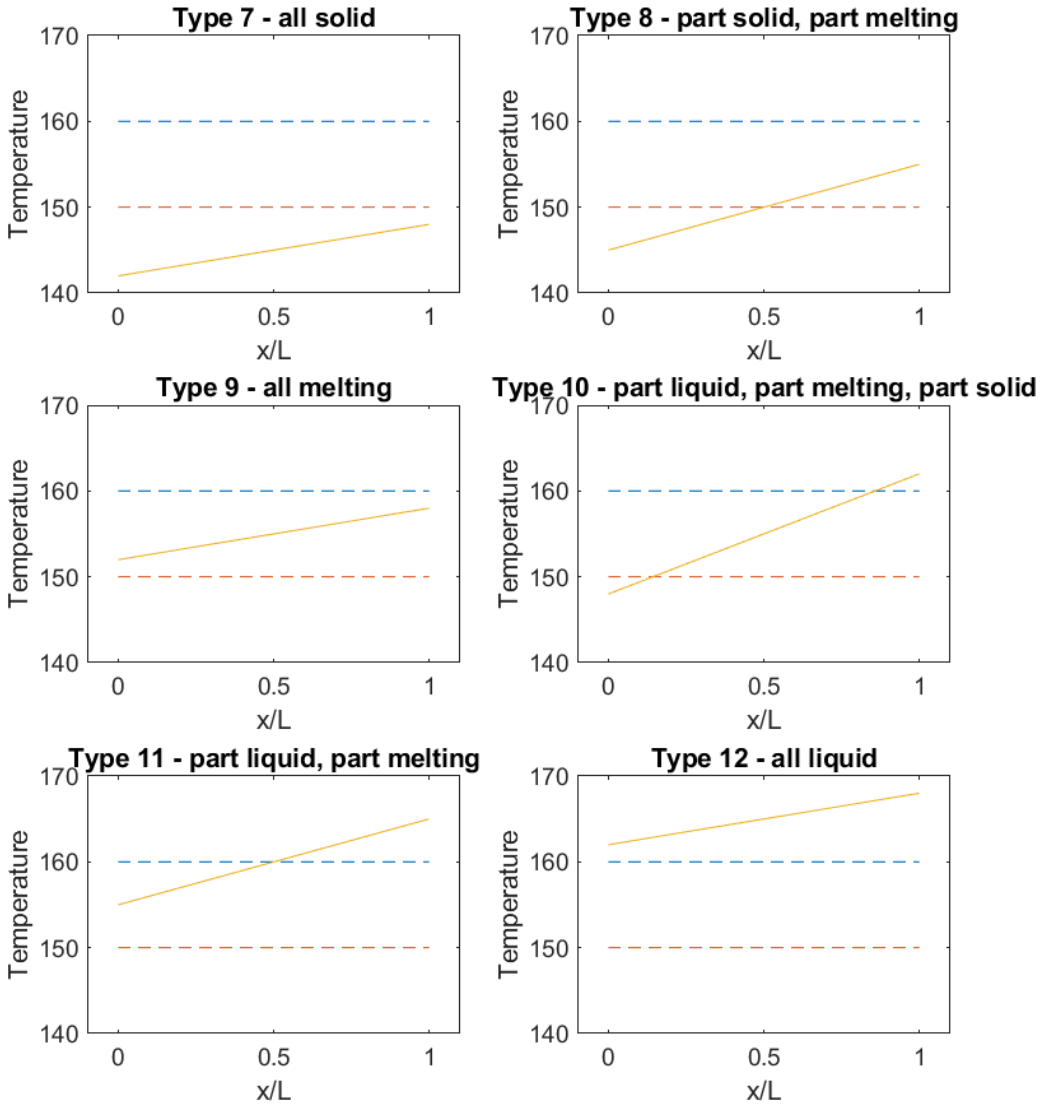


Figure 31: Element temperature distribution types 7-12

Table 5: Element force vectors

Type	$\{f_Q\}$
1	$\begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$
2	$\frac{h_{fs}AL}{2\left(T_1-T_2\right)^2}\begin{Bmatrix} \left(T_1-T_S\right)\left(T_1-2T_2+T_S\right) \\ \left(T_1-T_S\right)^2 \end{Bmatrix}$
3	$\frac{h_{fs}AL}{2}\begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$
4	$\frac{h_{fs}AL}{2\left(T_1-T_2\right)^2}\begin{Bmatrix} \left(T_L-T_S\right)\left(T_L-2T_2+T_S\right) \\ \left(T_L-T_S\right)\left(T_L-2T_1+T_S\right) \end{Bmatrix}$
5	$\frac{h_{fs}AL}{2\left(T_1-T_2\right)^2}\begin{Bmatrix} \left(T_2-T_L\right)^2 \\ \left(T_1-T_2\right)^2-\left(T_1-T_L\right)^2 \end{Bmatrix}$
6	$\begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$
7	$\begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$
8	$\frac{h_{fs}AL}{2\left(T_1-T_2\right)^2}\begin{Bmatrix} \left(T_2-T_S\right)^2 \\ \left(T_1-T_2\right)^2-\left(T_1-T_S\right)^2 \end{Bmatrix}$
9	$\frac{h_{fs}AL}{2}\begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$
10	$\frac{h_{fs}AL}{2\left(T_1-T_2\right)^2}\begin{Bmatrix} \left(T_L-T_S\right)\left(T_L-2T_2+T_S\right) \\ \left(T_L-T_S\right)\left(T_L-2T_1+T_S\right) \end{Bmatrix}$
11	$\frac{h_{fs}AL}{2\left(T_1-T_2\right)^2}\begin{Bmatrix} \left(T_1-T_L\right)\left(T_1-2T_2+T_L\right) \\ \left(T_1-T_L\right)^2 \end{Bmatrix}$
12	$\begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$

6.3 1-dimensional phase change with modified specific heat

An alternative model for the melting phenomenon can be found in [12]. It incorporates the latent heat as a different specific heat at constant pressure c_p within the melting range which is in between T_S and T_L. The latent heat h_{fs} leads to the following expression within the melting range

$$c_p* = \frac{h_{fs}}{T_L - T_S} + c_p \tag{58}$$

Using a simple if-alternative this can be incorporated into the melting code which was formulated in Matlab and compared to Abaqus results. The next section deals with the force vector approach and the specific heat approach.

6.4 Extension to cylindrical and spherical coordinates

In order to model 3-dimensional problems with the developed methods, different coordinate systems can be used and simplified using symmetry relations. Therefore a 3-dimensional calculation may only depend on the radial coordinate r. Fig. 32 shows a semi-infinite plane with a surface heat flux in its center. For a punctual heat flux such as a laser even a small pin can be modeled as a semi-infinite plane. Then this is evaluated in spherical coordinates using a symmetry plane which is placed in the same location as the outer surface. This calculation in spherical coordinates allows more problems to be analyzed than just the 1-dimensional Cartesian case. In the following subsection both cylindrical and spherical coordinates are used for the governing numerical equations. The conductivity matrix K and the mass matrix M can be seen in Chapter 3 (Eqs. 41 and 43)

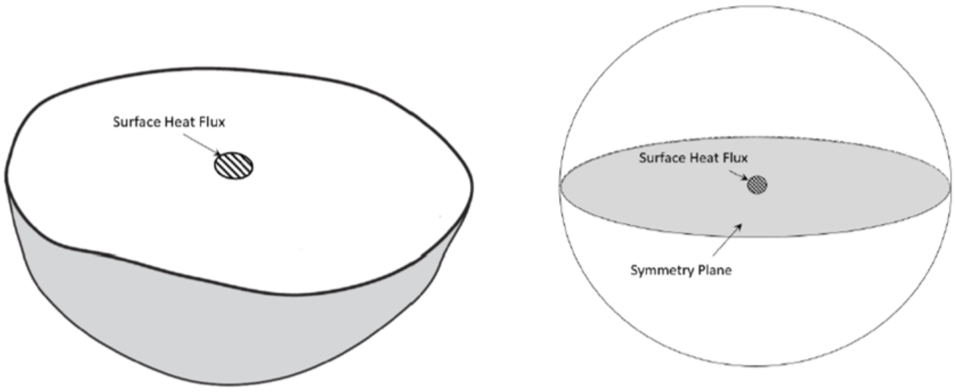


Figure 32: Localized heating of semi-infinite domain modelled as spherically symmetrical

6.5 1-dimensional melting problem

To evaluate the 1-dimensional phase change equations, transient heat transfer with melting is considered. A cylinder of radius r and length L is initially at temperature T_0. At time $t = 0$ the left end $x = 0$ temperature is set to T_1. The problem's dimensions and thermal properties are given in Table 6.

Table 6: Material properties and geometry for 1-dimensional cylinder melting

T_0	100 °C
T_1	200 °C
T_S	150 °C
T_L	160 °C
h_{fs}	3e6 W/(m^2 K)
k	0.195 W/(m K)
c_p	0.195 J/(kg K)
Δx	0.06 mm
L	6 mm
radius, r	1 mm
area, A	3.142 mm^2

The Matlab implementation of the latent heat force vector approach given in section 6.2 can be found in Appendices C and H. The computed temperature distri-

butions at times $t = 10$ s, 50 s and 100 s are shown in Figure 33 and are compared to results obtained using Abaqus. Although the overall temperature distributions correlate reasonably well, the Matlab code does not accurately track the location of the melt front. Further investigation is required to understand this discrepancy.

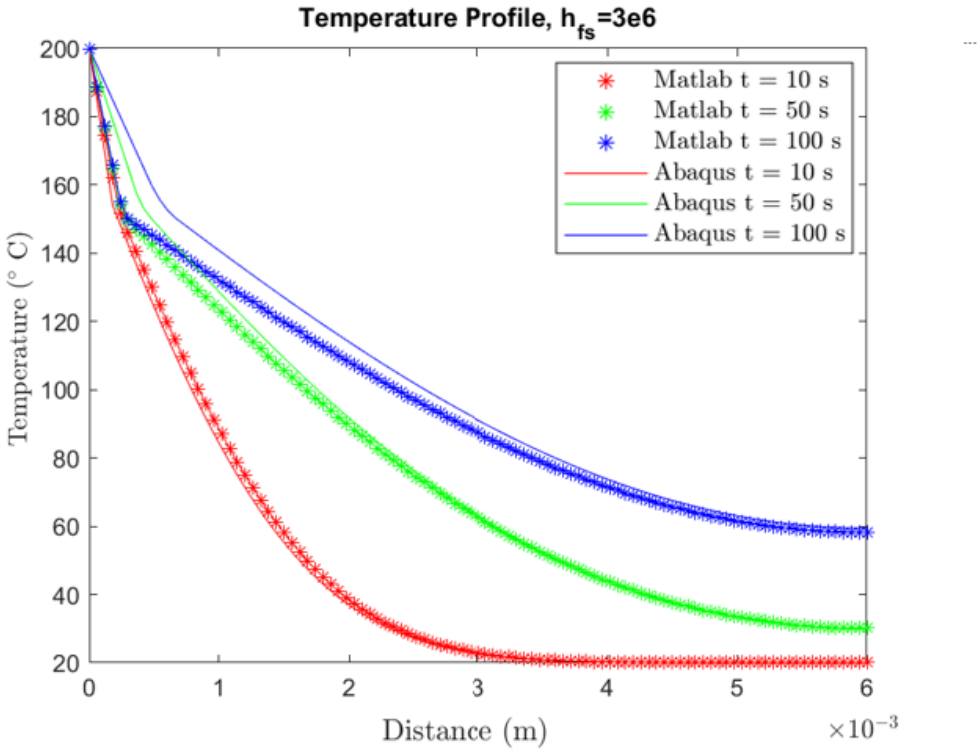


Figure 33: 1-dimensional melting with temperature boundary conditions

Now the specific heat approach is compared to the Abaqus results. The Abaqus simulation was done using Cartesian, cylindrical and spherical coordinates. The resulting temperature distributions for Cartesian, cylindrical and spherical coordinate systems over the depth of the cylinder are shown in Fig. 34 to 36. Additionally the growth of the melt front can be seen in Fig. 37. This depiction is similar to the freeze front which was shown in Fig. 24 to 29. The melt front grows

over the different finite elements. This leads to equidistant column heights which slowly grow at a slower rate because the penetration of the melt zone slows down. The used code can be found in Appendix I. The Abaqus data can be shared upon request.

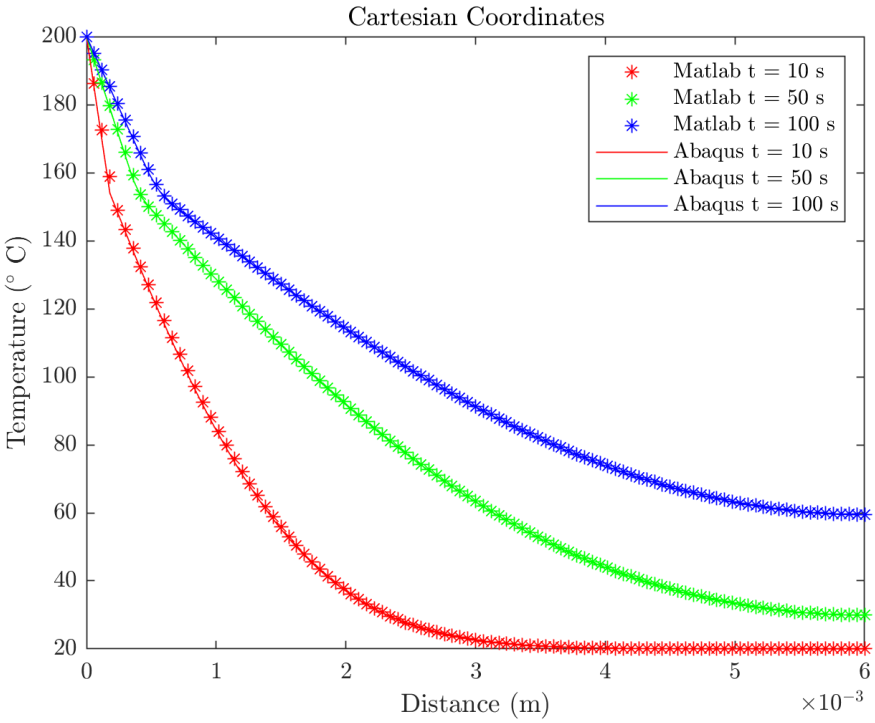


Figure 34: 1-dimensional melting in Cartesian coordinate system

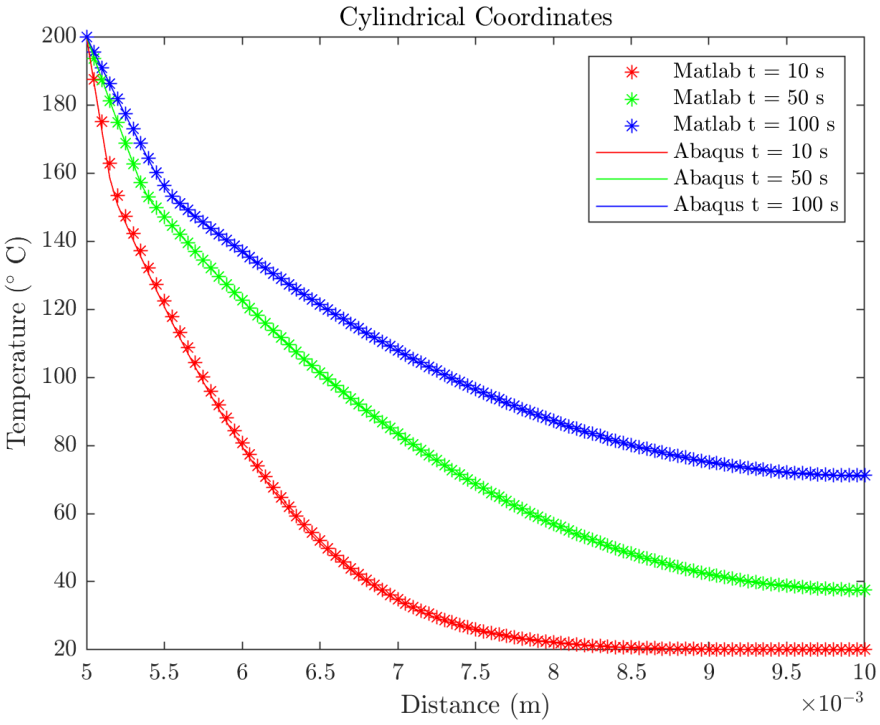


Figure 35: 1-dimensional melting in cylindrical coordinate system

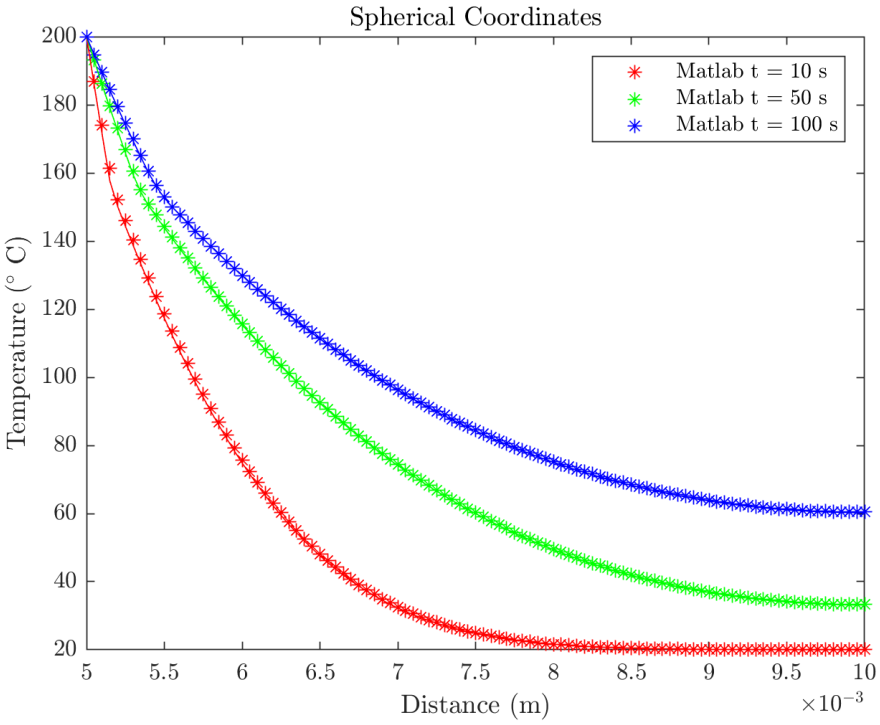


Figure 36: 1-dimensional melting in spheical coordinate system

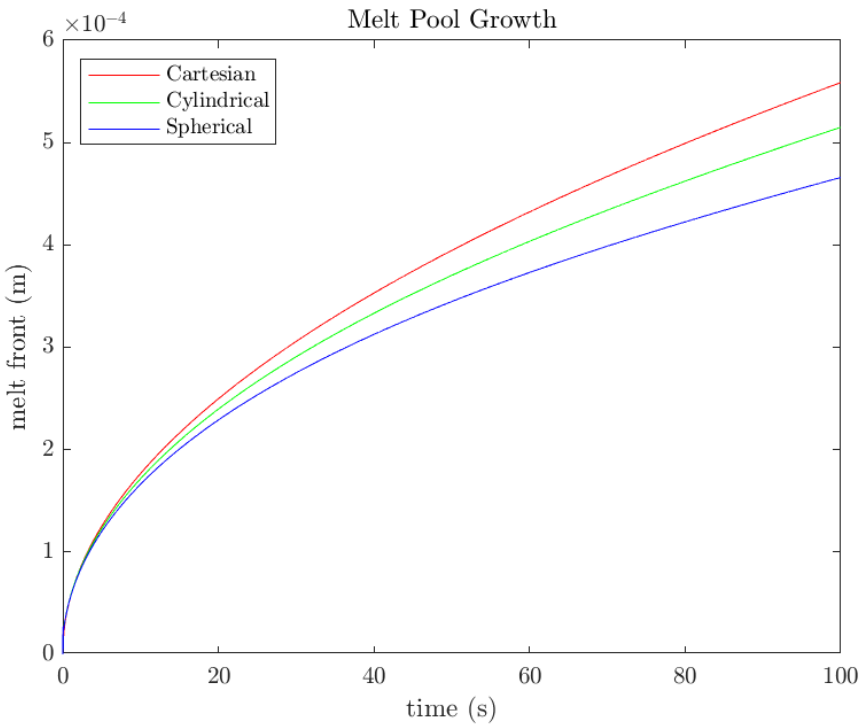


Figure 37: 1-dimensional melting in different coordinate systems

CHAPTER 7

Conclusion

Different heating problems were simulated using commercial software and self-written code. Various numerical approximation schemes were used and compared. Finally a finite element code for melting was compared to commercial software for two different latent heat models. A 1-dimensional Cartesian model was adapted for spherically symmetric melting, where a melt zone grows radially from the center of the domain. This can solve 3-dimensional problems with spherical symmetry. A laser melting a pin can be assumed to be a semi-infinite sphere. Such a solution is a computationally efficient estimate of melt zone growth in the vicinity of a focused heat source on the surface of a semi-infinite domain. This modeling capability could prove to be valuable in studying additive manufacturing processes.

LIST OF REFERENCES

[1] M. H. Sadd, *Elasticity: theory, applications, and numerics.* Academic Press, 2014.

[2] A. F. Mills, *Heat Transfer.* R.R. Donnelly Sons, 1992.

[3] D. L. Logan, *A First Course in the Finite Element Method.* Brooks/Cole, 2012.

[4] [Online]. Available: https://abaqus-docs.mit.edu/2017/English/SIMACAEBMKRefMap/simabmk-c-freezingofsolid.htm

[5] W.-J. Zhang, Z.-Y. Liu, Z.-L. Liu, and L.-C. Cai, "Melting curves and entropy of melting of iron under earth's core conditions," *Physics of the Earth and Planetary Interiors*, vol. 244, pp. 69–77, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031920114002222

[6] M. Yaakoubi, M. Kchaou, and F. Dammak, "Simulation of the thermomechanical and metallurgical behavior of steels by using abaqus software," *Computational Materials Science*, vol. 68, pp. 297–306, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927025612005824

[7] R. Mabrouk, H. Dhahri, H. Naji, S. Hammouda, and Z. Younsi, "Lattice boltzmann simulation of forced convection melting of a composite phase change material with heat dissipation through an open-ended channel," *International Journal of Heat and Mass Transfer*, vol. 153, p. 119606, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0017931019343157

[8] J. Holman, *Heat Transfer.* Mcgraw-Hill, 2010.

[9] P. Stephan, K. Schaber, K. Stephan, and F. Mayinger, *Thermodynamik: Grundlagen und technische Anwendungen Band 1: Einstoffsysteme.* Springer Berlin Heidelberg, 2013.

[10] G. M. D., *Advanced Engineering Mathematics.* Prentice-Hall, 1998.

[11] A. Lazaridis, "Numerical solution of the multidimensional solidification (or melting) problem," *International Journal of Heat and Mass Transfer*, vol. 13, pp. 1459–1477, 1970.

[12] W. Ogoh and D. Groulx, "Stefan's problem: Validation of a one-dimensional solid-liquid phase change heat transfer process," in *Comsol Conference 2010*, 2010.

APPENDIX A

Matlab script for 1-dimensional finite difference method in Cartesian coordinates

```
% Heat Conduction Equation in one dimension
% Solves dT/dt = a *( d^2T/dx^2 + 2 dT/(dx^2 *r) )

clc;
close all;
clear all;

%dimensions
N = 51;

Dx=0.0005; % step size
Nx=0.025;
X=0:Dx:Nx;
k = 0.627; % thermal conductivity in W / (m K)
rho = 993.95; %denisty in kg/m^3
cp = 4178.5; %specific heat at const. press. in J / (kg K)
a=1; % arbitrary thermal diffusivity
a = k / (rho*cp);
h = 1200; %W/ m deg C
Tinf = 95;
Bi = h * Dx / k;

%boundary conditions
T(1:N) = 5 ;
T(N) = 95;  %temp bound. cond.
T(N) = (2*T(N-1) + Bi*Tinf)/(2+Bi);


%%
%initial condition
Tmax=max(T);

DT = 0.05;
%DT = DR^2/(2*a); % time step
M = 20000; % maximum number of allowed iteration

%finite difference scheme

fram=0;
Ncount=0;
loop=1;

while loop==1;
   ERR=0;
   T_old = T;

for i = 2:N-1

   Residue=(DT*((T_old(i+1)-2*T_old(i)+T_old(i-1))/Dx^2 ...
                     + 2*(T_old(i+1) - T_old(i))/(Dx^2 * i))...
                     + T_old(i))-T(i);
   T_old(1) = T_old(2);
   ERR = ERR+abs(Residue);
  T(i) = T(i)+Residue*a;
  T(1) = T(2);
  T(N) = (T(N-1) + T_old(N) + Bi*Tinf)/(2+Bi);

end
if(ERR>=0.000001*Tmax)  % allowed error limit is 1%
```

```
% of maximum temperature
    Ncount=Ncount+1;
         if (mod(Ncount,50)==0) % displays movie
         % frame every 50 time steps
              fram=fram+1;
              plot(T);
              ylim([5,95])
              axis([1 N 20 100])
              h=gca;
              get(h,'FontSize')
              set(h,'FontSize',12)
              xlabel('radial distance from 0 to 0.025 m',...
              'fontSize',12);
              ylabel('temperature in deg C','fontSize',...
              12);
              title('temperature distribution in egg ',...
              'fontsize',12);
              fh = figure(1);
             set(fh, 'color', 'white');
            F=getframe;
         end

 % if solution do not converges
 % in M time steps

    if(Ncount>M)
        loop=0;
        disp(['solution does not reach...
        steady state in  ',num2str(M),...
            'time steps'])
    end

 % if solution converges within x time ste
 % ps

else
    loop=0;
    disp(['solution reaches steady state in ',...
    num2str(Ncount) ,'time steps'])
end
end

% display a movie of heat diffusion
 movie(F,fram,1);
 disp(['time elapsed:',num2str(Ncount*DT) ,'s'])
```

APPENDIX B

Matlab script for 1-dimensional finite difference method in spherical coordinates

```
% Spherical Heat Conduction Equation in one dimension
% Solves dT/dt = a *( d^2T/dr^2 + 2 dT/(dr^2 *r) )

clc;
close all;
clear all;

%dimensions
N = 51;

DR=0.0005; % step size
Nr=0.025;
X=0:DR:Nr;
k = 0.627; % thermal conductivity in W / (m K)
rho = 993.95; %denisty in kg/m^3
cp = 4178.5; %specific heat at const. press. in J / (kg K)
a=1; % arbitrary thermal diffusivity
a = k / (rho*cp);
h = 1200; %W/ m deg C
Tinf = 95;
Bi = h * DR / k;

%boundary conditions
T(1:N) = 5 ;
T(N) = 95;  %temp bound. cond.
T(N) = (2*T(N-1) + Bi*Tinf)/(2+Bi);

%for cart., works: (2*T(N-1) + Bi*Tinf)/(2+Bi);
% doesnt work:
%(T(N-1) + h*DR/k*Tinf)/(1 + h*DR/k); %convective bound cond.
% k * dT/dx = h * (T-Tinf)
%dT/dx = T(N) - T(N-1) / dx or T(N-1) - T(N) / dx
%h * (T(N) - Tinf)
%T(N) = (T(N-1) - h*dx/k*Tinf)/(1 - h*dx/k)
%%
%initial condition
Tmax=max(T);

DT = 0.05;
%DT = DR^2/(2*a); % time step
M = 20000; % maximum number of allowed iteration

%finite difference scheme

fram=0;
Ncount=0;
loop=1;

while loop==1;
   ERR=0;
   T_old = T;

for i = 2:N-1

   Residue=(DT*((T_old(i+1)-2*T_old(i)+T_old(i-1))/DR^2 ...
                     + 2*(T_old(i+1) - T_old(i))/(DR^2 * i))...
                     + T_old(i))-T(i);
   T_old(1) = T_old(2);
   ERR = ERR+abs(Residue);
```

```
  T(i) = T(i)+Residue*a;
  T(1) = T(2);
  T(N) = (T(N-1) + T_old(N) + Bi*Tinf)/(2+Bi);
  %old: (T(N-1) - h*DR/k*Tinf)/(1 - h*DR/k); %for conv. bound. cond.
end
if(ERR>=0.000001*Tmax)  % allowed error limit is
% 1% of maximum temperature
    Ncount=Ncount+1;
         if (mod(Ncount,50)==0)
         % displays movie frame every 50 time steps
              fram=fram+1;
              plot(T);
              ylim([5,95])
              axis([1 N 20 100])
              h=gca;
              get(h,'FontSize')
              set(h,'FontSize',12)
%               colorbar('location','eastoutside','fontsize',12);
              xlabel('radial distance from ...
              0 to 0.025 m','fontSize',12);
              ylabel('temperature in deg C','fontSize',12);
              title('temperature distribution in egg ',...
              'fontsize',12);
              fh = figure(1);
             set(fh, 'color', 'white');
            F=getframe;
         end

 %if solution does not converge in
 % 2000 time steps

    if(Ncount>M)
        loop=0;
        disp(['solution do not...
        reach steady state in  ',num2str(M),...
            'time steps'])
    end

 % if solution converges within 2000 time steps
else
    loop=0;
    disp(['solution reaches steady state in ',...
    num2str(Ncount) ,'time steps'])
end
end
% display a movie of heat diffusion
movie(F,fram,1);
 disp(['time elapsed:',num2str(Ncount*DT) ,'s'])
```

APPENDIX C

Matlab script for validation of 1-dimensional finite element methods

```
% 1-D conduction
clc; clear all; close all; format compact
set(groot,'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
for coord_type=1:3   % (1- Cartesian, 2-Cylindrical, 3- Spherical)
    %% set constants

    h = 40;         % W/m^2 K
    rho = 2600;     % kg/m^3
    c = 800;        % J/kg K
    k = 1;          % W/m K

    %% define mesh
    x1=0.05;    % left node coordinate(m)
    x2=.1;      % right node coordinate (m)
    L = x2-x1; % m
    N=21;        % number of nodes;
    dx=L/(N-1);
    xfea=x1:dx:x2;

    %% set BC
    T1=20;
    T2=100;

    %% calculate element stiffness matrix and assemble
    global stiff. mat.
    if coord_type==1
        A=1; % m^2
        Ke = A*k/dx*[1,-1;-1,1];
        K = zeros(N);
        for i = 1:N-1
            K(i,i) = K(i,i)+ Ke(1,1);
            K(i+1,i+1) = K(i+1,i+1) + Ke(2,2);
            K(i,i+1) = Ke(1,2);
            K(i+1,i) = Ke(2,1);
        end
    elseif coord_type==2
        K = zeros(N);
        for i = 1:N-1
            Ke = 2*pi*k*(xfea(i)+xfea(i+1))/(2*dx)*[1,-1;-1,1];
            K(i,i) = K(i,i)+ Ke(1,1);
            K(i+1,i+1) = K(i+1,i+1) + Ke(2,2);
            K(i,i+1) = Ke(1,2);
            K(i+1,i) = Ke(2,1);
        end
    else
        K = zeros(N);
        for i = 1:N-1
            Ke=4*pi*k*(xfea(i)^2+xfea(i)*xfea(i+1)+xfea(i+1)^2)/ ...
            (3*dx)*[1,-1;-1,1];
            K(i,i) = K(i,i)+ Ke(1,1);
            K(i+1,i+1) = K(i+1,i+1) + Ke(2,2);
            K(i,i+1) = Ke(1,2);
            K(i+1,i) = Ke(2,1);
        end
    end
    %% set temperature BCs solve equations
```

```
    F=zeros(N-2,1);
    F(1)=-K(2,1)*T1;
    F(N-2)=F(N-2)-K(N-1,N)*T2;
    Kred=K(2:N-1,2:N-1);
    Tred=Kred\F;
    Tfea=[T1;Tred;T2];
    K*Tfea;
    %% compare to exact
    xexact=linspace(x1,x2,50);
    if coord_type==1
        Texact=T1+(T2-T1)*(xexact-x1)/L;
    elseif coord_type==2
        Texact=T1+(T2-T1)*log(xexact/x1)/log(x2/x1);
    else
        Texact=T1+(T2-T1)*(x2./xexact).*(xexact-x1)/L;
    end
    %
    h=figure;
    plot(xfea,Tfea,'*',xexact,Texact,'-')
    xlabel('Position (m)')
    ylabel('Temperature (C)')
    axis([x1-.01 x2+.01 T1-5 T2+5])
    legend('FEA','Exact','Location','NorthWest')
    if coord_type==1
        filename='conduction_1D_cartesian';
        title('Cartesian Coordinates')
    elseif coord_type==2
        filename='conduction_1D_cylindrical';
        title('Cylindrical Coordinates')
    else
        filename='conduction_1D_spherical';
        title('Spherical Coordinates')
    end
    saveas(h,filename,'png');
end


% 1-D conduction with outer wall convection
clc; clear all; close all; format compact
set(groot,'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
for coord_type=1:3  % (1- Cartesian, 2-Cylindrical, 3- Spherical)
    clear T
    %% set constants
    if coord_type==1
        T0 = 20;      % deg C
        Tinf = 100;   % deg C
        h = 40;       % W/m^2 K
        rho = 2600;   % kg/m^3
        c = 800;      % J/kg K
        k = 1;        % W/m K
    elseif coord_type==2
        T0 = 20;      % deg C
        Tinf = 100;   % deg C
        h = 40;       % W/m^2 K
        rho = 2600;   % kg/m^3
        c = 800;      % J/kg K
        k = 1;        % W/m K
    else
        T0 = 5;      % deg C
        Tinf = 95;   % deg C
        h =  1200;   % W/m^2 K
        rho = 993;   % kg/m^3
        c =  4178;   % J/kg K
        k = 0.627;   % W/m K
```

```
end
%% define mesh
if coord_type==1
    % for Cartesian, use Mills problem
    x1=0;         % left node coordinate(m)
    x2=.08;       % right node coordinate (m)
    L = x2-x1;    % mesh length (m)
    N=5;          % number of nodes;
    dx=L/(N-1);   % element length (m)
    xfea=x1:dx:x2;
    %
elseif coord_type==2
    x1=0.05;     % left node coordinate(m)
    x2=.1;       % right node coordinate (m)
    dz=1;        % unit height in z-direction (m)
    L = x2-x1;   % m
    N=51; %21;       % number of nodes;
    dx=L/(N-1); % element length (m)
    xfea=x1:dx:x2;
else
    % for Spherical, use boiling egg problem
    x1=0;         % left node coordinate(m)
    x2=.025;      % right node coordinate (m)
    L = x2-x1;    % m
    N=21;         % number of nodes;
    dx=L/(N-1);
    xfea=x1:dx:x2;
end

%% set Initial condition
if coord_type==1
    T(1,:) = T0*ones(N,1);
elseif coord_type==2
    T(1,:) = T0*ones(N,1);
else
    T(1,:) = T0*ones(N,1);
end

%% calculate element stiffness matrix and
% assemble global stiff. mat.
if coord_type==1
    A=1; % m^2
    Ke = A*k/dx*[1,-1;-1,1];
    Kh = h*A*[0 0; 0 1];
    K = zeros(N);
    % first, add K for last element
    K(N-1:N,N-1:N)=Kh;
    for i = 1:N-1
        K(i,i) = K(i,i)+ Ke(1,1);
        K(i+1,i+1) = K(i+1,i+1) + Ke(2,2);
        K(i,i+1) = Ke(1,2);
        K(i+1,i) = Ke(2,1);
    end
    m = c*rho*A*dx/6 *[2,1;1,2];
    M = zeros(N);
    for i = 1:N-1
        M(i,i) = M(i,i)+ m(1,1);
        M(i+1,i+1) = M(i+1,i+1) + m(2,2);
        M(i,i+1) = m(1,2);
        M(i+1,i) = m(2,1);
    end
elseif coord_type==2
    K = zeros(N);
    A=2*pi*x2*dz;
    % first, add K for last element
    % (terms are per unit height, dz)
```

```
        Kh = 2*pi*x2*h*[0 0; 0 1];
        K(N-1:N,N-1:N)=Kh;
        for i = 1:N-1
            Ke = 2*pi*k*(xfea(i)+xfea(i+1))/(2*dx)*[1,-1;-1,1];
            K(i,i) = K(i,i)+ Ke(1,1);
            K(i+1,i+1) = K(i+1,i+1) + Ke(2,2);
            K(i,i+1) = Ke(1,2);
            K(i+1,i) = Ke(2,1);
        end
        M = zeros(N);
        for i = 1:N-1
            r1=xfea(i);
            r2=xfea(i+1);
            m(1,1) = 2*pi*c*rho*(r2-r1)*(r1/4+r2/12);
            m(1,2) = 2*pi*c*rho*(r2-r1)*(r1/12+r2/12);
            m(2,1) = m(1,2);
            m(2,2) = 2*pi*c*rho*(r2-r1)*(r1/12+r2/4);
            M(i,i) = M(i,i)+ m(1,1);
%             m(1,1) = 2*pi*c*rho*(r2-r1)^3*(r1/4+r2/12);
%             m(1,2) = 2*pi*c*rho*(r2-r1)^3*(r1/12+r2/12);
%             m(2,1) = m(1,2);
%             m(2,2) = 2*pi*c*rho*(r2-r1)^3*(r1/12+r2/4);
%             M(i,i) = M(i,i)+ m(1,1);
            M(i+1,i+1) = M(i+1,i+1) + m(2,2);
            M(i,i+1) = M(i,i+1) + m(1,2);
            M(i+1,i) = M(i+1,i) + m(2,1);
        end
    else
        K = zeros(N);
        % first, add K for last element
        Kh = 4*pi*x2^2*h*[0 0; 0 1];
        K(N-1:N,N-1:N)=Kh;
        for i = 1:N-1
            Ke=4*pi*k*(xfea(i)^2+xfea(i)*xfea(i+1)+xfea(i+1)^2)/ ...
            (3*dx)*[1,-1;-1,1];
            K(i,i) = K(i,i)+ Ke(1,1);
            K(i+1,i+1) = K(i+1,i+1) + Ke(2,2);
            K(i,i+1) = K(i,i+1) + Ke(1,2);
            K(i+1,i) = K(i+1,i) + Ke(2,1);
        end
        K;
        M = zeros(N);
        for i = 1:N-1
            r1=xfea(i);
            r2=xfea(i+1);
            m(1,1) = 4*pi*c*rho*(r2-r1)*(r1^2/5+r1*r2/10+r2^2/30);
            m(1,2) = 4*pi*c*rho*(-3*r1^5+5*r1^4*r2-5*r1*r2^4+...
            3*r2^5)/(60*(r2-r1)^2);
            m(2,1) = m(1,2);
            m(2,2) = 4*pi*c*rho*(r2-r1)*(r1^2/30+r1*r2/10+r2^2/5);
            M(i,i) = M(i,i)+ m(1,1);
            M(i+1,i+1) = M(i+1,i+1) + m(2,2);
            M(i,i+1) = M(i,i+1) + m(1,2);
            M(i+1,i) = M(i+1,i) + m(2,1);
        end
        M;
    end

    %% calculate element force vector
    % and assemble global force vector
    if coord_type==1
        f = [0,0];
        F = zeros(N,1);
        F(N) = A*h*Tinf;
    elseif coord_type==2
```

```
%         % Temp BC at inner and outer radius
%         T1=20;
%         T2=100;
%         F=zeros(N-2,1);
%         F(1)=-K(2,1)*T1;
%         F(N-2)=F(N-2)-K(N-1,N)*T2;
%         Kred=K(2:N-1,2:N-1);
%         Tred=Kred\F;
%         Tfea=[T1;Tred;T2];
        f = [0,0];
        F = zeros(N,1);
        F(N) = A*h*Tinf;
    else
        f = [0,0];
        F = zeros(N,1);
        F(N) = 4*pi*x2^2*h*Tinf;
    end

    %% calculate matrices for iteration time stepping
    if coord_type==1
        dt=60;      % time step increment (s)
        % set beta (parameter for weighting explicit
        % and implicit calculation
        beta = .5;  % Crank-Nicholson: 0.5
        A1 = M/dt + beta*K;
        B = M/dt - (1-beta)*K; % same as A for Crank-Nicholson
        % %solve for t_i (stored in matrix T(t,i)
        % with t timesteps and i nodes
        tp=zeros(20,1);
        tp(1)=0;
        for t = 1:((70*60)/dt-1)
            tp(t+1)=tp(t)+dt;
            T(t+1,:) = A1\(B * T(t,:)' + F); % '
        end
    elseif coord_type==2
        dt=1;       % time step increment (s)
        tmax=3600;       % final time (s)
        % set beta (parameter for weighting explicit
        % and implicit calculation
        beta = .5;  % Crank-Nicholson: 0.5
        A1 = M/dt + beta*K;
        B = M/dt - (1-beta)*K; % same as A
        % for Crank-Nicholson
        % %solve for t_i (stored in matrix T(t,i)
        % with t timesteps and i nodes
%         tp=zeros(ceil(tmax/dt),1);
        tp(1)=0;
        for t = 1:tmax/dt
            tp(t+1)=tp(t)+dt;
            T(t+1,:) = A1\(B * T(t,:)' + F);
%             disp(['Time. T1, T2 = ' num2str(tp(t)),...
 ' num2str(T(t+1,1)) ', ' num2str(T(t+1,end)) ])
        end


    else
        dt=1;      % time step increment (s)
        % set beta (parameter for weighting explicit
        % and implicit calculation
        beta = .5;  % Crank-Nicholson: 0.5
        A1 = M/dt + beta*K;
        B = M/dt - (1-beta)*K; % same as A for Crank-Nicholson
        % solve for t_i (stored in matrix T(t,i)
        % with t timesteps and i nodes
        tp=zeros(1000/dt,1);
        tp(1)=0;
```

```
        for t = 1:1000/dt
            tp(t+1)=tp(t)+dt;
            T(t+1,:) = A1\(B * T(t,:)' + F); %'
        end
    end
%     T
    %% compare to exact
    if coord_type==1
        exact=[0 20
            3.47  46.32
            6.93  53.30
            10.4  57.68
            13.87 60.85
            17.33 63.32
            20.80 65.33
            24.27 67.01
            27.72 68.45
            31.20 69.71
            34.67 70.82
            38.13 71.81
            41.60 72.72
            45.07 73.54
            48.53 74.31
            52.00 75.02
            55.47 75.69
            58.93 76.32
            62.40 76.92
            65.87 77.50
            69.33 78.05];
        t_exact=60*exact(:,1);
        T_exact=exact(:,2);
        h=figure;
        plot(tp,T(:,end),'.',t_exact,T_exact,'-')
        xlabel('Time (s)')
        ylabel('Temperature (C)')
        legend('FEA','Exact','Location','NorthWest')
        axis([0 4500 10 90])
    elseif coord_type==2
        h=figure;
        exact=[0.                20.                20.
              100.               20.                40.4818
              200.               20.0093            46.8965
              300.               20.1234            51.2019
              400.               20.4915            54.4783
              500.               21.1787            57.1297
              600.               22.1691            59.3574
              700.               23.4109            61.2787
              800.               24.8466            62.9686
              900.               26.4244            64.4777
                1.E+03           28.0996            65.8445
                1.1E+03          29.8374            67.0965
                1.2E+03          31.6109            68.2544
                1.3E+03          33.3991            69.3345
                1.4E+03          35.1863            70.3492
                1.5E+03          36.9607            71.3082
                1.6E+03          38.7135            72.2192
                1.7E+03          40.4383            73.0881
                1.8E+03          42.1305            73.9196
                1.9E+03          43.787             74.7176
                2.E+03           45.4054            75.4851
                2.1E+03          46.9846            76.2246
                2.2E+03          48.5236            76.9383
                2.3E+03          50.0223            77.6278
                2.4E+03          51.4804            78.2943
                2.5E+03          52.8986            78.9394
                2.6E+03          54.2772            79.5641
                2.7E+03          55.617             80.1693
```

```
                2.8E+03          56.9187          80.7558
                2.9E+03          58.183           81.3242
                3.E+03           59.4108          81.8755
                3.1E+03          60.6031          82.4101
                3.2E+03          61.7608          82.9287
                3.3E+03          62.8848          83.4318
                3.4E+03          63.9758          83.9199
                3.5E+03          65.035           84.3934
                3.6E+03          66.0631          84.853     ];
        t_exact=exact(:,1);
        T_exact_inner=exact(:,2);
        T_exact_outer=exact(:,3);
        plot(tp,T(:,1),'b.',t_exact,T_exact_inner,'b-', ...
            tp,T(:,end),'r.',t_exact,T_exact_outer,'r-')
        xlabel('Time (s)')
        ylabel('Temperature (C)')
        legend('FEA Inner Center','Exact Inner Center',...
        'FEA Outer Surface','Exact Outer Surface',...
        'Location','NorthWest')
        axis([0 3600 10 110])
    else
        h=figure;
        for i=2:size(tp,1)
            if T(i-1,1)<70 && T(i,1)>=70
                t70=tp(i);
            end
        end
        disp([tp T(:,1)])
        tp;
        T;
        plot(tp,T(:,1),'b.',tp,T(:,end),'r.')
        xlabel('Time (s)')
        ylabel('Temperature (C)')
        legend('Center','Surface','Location','NorthWest')
        text(500,20,['time step = ' num2str(dt)])
        text(500,15,['time to reach 70 deg = ' num2str(t70) ' sec'])
        axis([0 1000 0 110])
    end
    if coord_type==1
        filename='conduction_1D_convection_cartesian';
        title('Cartesian Coordinates - Mill''s problem')
    elseif coord_type==2
        filename='conduction_1D_convection_cylindrical';
        title('Cylindrical Coordinates - Convection BC''s')
    else
        filename='conduction_1D_convection_spherical';
        title('Spherical Coordinates - Boiling egg problem')
    end
    saveas(h,filename,'png');
end


% 1-D conduction with outer wall convection
clc; clear all; close all; format compact
set(groot,'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
for coord_type=1:3  % (1- Cartesian, 2-Cylindrical, 3- Spherical)
    clear T
    %% set constants
    if coord_type==1
        % PLA
        qA=5.;          % Watt
        T0 = 20;      % deg C
        k = 0.195;    % W/m K
        c = 2060;     % cp = J/kg K
```

```
    rho = 1200;  % kg/m^3
elseif coord_type==2
    qA=100000*(2*pi*.05*1.);  % Watt (unit height in z-direction)
    T0 = 20;              % deg C
    rho = 2600;           % kg/m^3
    c = 800;              % J/kg K
    k = 100;                % W/m K
else
    qA=100000*(4*pi*.05^2);  % Watt (unit height in z-direction)
    T0 = 20;              % deg C
    rho = 2600;           % kg/m^3
    c = 800;              % J/kg K
    k = 100;                % W/m K
end
%% define mesh
if coord_type==1
    % PLA "peg" geometry
    x1=0;          % left node coordinate(m)
    x2=.006;       % right node coordinate (m)
    L = x2-x1;     % mesh length (m)
    N=31;          % number of nodes;
    dx=L/(N-1);  % element length (m)
    xfea=x1:dx:x2;
elseif coord_type==2
    x1=0.05;     % left node coordinate(m)
    x2=.1;       % right node coordinate (m)
    dz=1;        % unit height in z-direction (m)
    L = x2-x1;  % m
    N=21;        % number of nodes;
    dx=L/(N-1); % element length (m)
    xfea=x1:dx:x2;
else
    %
    x1=0.05;      % left node coordinate(m)
    x2=.1;        % right node coordinate (m)
    L = x2-x1;    % m
    N=21;         % number of nodes;
    dx=L/(N-1);  % element length (m)
    xfea=x1:dx:x2;
end

%% set Initial condition
if coord_type==1
    T(1,:) = T0*ones(N,1);
elseif coord_type==2
    T(1,:) = T0*ones(N,1);
else
    T(1,:) = T0*ones(N,1);
end

%% calculate element stiffness matrix
% and assemble global stiff. mat.
if coord_type==1
    A=pi*.001^2; % m^2
    Ke = A*k/dx*[1,-1;-1,1];
    K = zeros(N);
    for i = 1:N-1
        K(i,i) = K(i,i)+ Ke(1,1);
        K(i+1,i+1) = K(i+1,i+1) + Ke(2,2);
        K(i,i+1) = K(i,i+1) + Ke(1,2);
        K(i+1,i) = K(i+1,i) + Ke(2,1);
    end
    K;
    m = c*rho*A*dx/6 *[2,1;1,2];
    M = zeros(N);
    for i = 1:N-1
```

```
        M(i,i) = M(i,i)+ m(1,1);
        M(i+1,i+1) = M(i+1,i+1) + m(2,2);
        M(i,i+1) = M(i,i+1) + m(1,2);
        M(i+1,i) = M(i+1,i) + m(2,1);
    end
    M;
elseif coord_type==2
    K = zeros(N);
    A=2*pi*x2*dz;
    for i = 1:N-1
        Ke = 2*pi*k*(xfea(i)+xfea(i+1))/(2*dx)*[1,-1;-1,1];
        K(i,i) = K(i,i)+ Ke(1,1);
        K(i+1,i+1) = K(i+1,i+1) + Ke(2,2);
        K(i,i+1) = Ke(1,2);
        K(i+1,i) = Ke(2,1);
    end
    M = zeros(N);
    for i = 1:N-1
        r1=xfea(i);
        r2=xfea(i+1);
        m(1,1) = 2*pi*c*rho*(r2-r1)*(r1/4+r2/12);
        m(1,2) = 2*pi*c*rho*(r2-r1)*(r1/12+r2/12);
        m(2,1) = m(1,2);
        m(2,2) = 2*pi*c*rho*(r2-r1)*(r1/12+r2/4);
        M(i,i) = M(i,i)+ m(1,1);
        M(i+1,i+1) = M(i+1,i+1) + m(2,2);
        M(i,i+1) = M(i,i+1) + m(1,2);
        M(i+1,i) = M(i+1,i) + m(2,1);
    end
else
    K = zeros(N);
    for i = 1:N-1
        Ke=4*pi*k*(xfea(i)^2+xfea(i)*xfea(i+1)+...
        xfea(i+1)^2)/(3*dx)*[1,-1;-1,1];
        K(i,i) = K(i,i)+ Ke(1,1);
        K(i+1,i+1) = K(i+1,i+1) + Ke(2,2);
        K(i,i+1) = K(i,i+1) + Ke(1,2);
        K(i+1,i) = K(i+1,i) + Ke(2,1);
    end
    K;
    M = zeros(N);
    for i = 1:N-1
        r1=xfea(i);
        r2=xfea(i+1);
        m(1,1) = 4*pi*c*rho*(r2-r1)*(r1^2/5+r1*r2/10+r2^2/30);
        m(1,2) = 4*pi*c*rho*(-3*r1^5+5*r1^4*r2-5*r1*r2^4+...
        3*r2^5)/(60*(r2-r1)^2);
        m(2,1) = m(1,2);
        m(2,2) = 4*pi*c*rho*(r2-r1)*(r1^2/30+r1*r2/10+r2^2/5);
        M(i,i) = M(i,i)+ m(1,1);
        M(i+1,i+1) = M(i+1,i+1) + m(2,2);
        M(i,i+1) = M(i,i+1) + m(1,2);
        M(i+1,i) = M(i+1,i) + m(2,1);
    end
    M;
end

%% calculate element force vector and
assemble global force vector
if coord_type==1
    f = [0,0];
    F = zeros(N,1);
    F(1) = qA;  % flux = (power (W) / area (m^2))) * area (m^2)
elseif coord_type==2
    f = [0,0];
    F = zeros(N,1);
```

```
    F(1) = qA;
else
    f = [0,0];
    F = zeros(N,1);
    F(1) = qA;
end

%% calculate matrices for iteration time stepping
if coord_type==1


    %% set temperature BCs solve equations
    dt=.02;     % time step increment (s)
    tmax=2;      % final time (s)
    % set beta (parameter for weighting explicit
    and implicit calculation
    beta = .5;  % Crank-Nicholson: 0.5
    A1 = M/dt + beta*K;
    B = M/dt - (1-beta)*K; % same as A for Crank-Nicholson
    % %solve for t_i (stored in matrix T(t,i)
    with t timesteps and i nodes
    tp=zeros(floor(tmax/dt),1);
    tp(1)=0;
    for t = 1:floor(tmax/dt)
        tp(t+1)=tp(t)+dt;
        T(t+1,:) = A1\(B * T(t,:)' + F); %'
    end
elseif coord_type==2
    dt=.02;     % time step increment (s)
    tmax=10;      % final time (s)
    % set beta (parameter for weighting
    explicit and implicit calculation
    beta = .5;  % Crank-Nicholson: 0.5
    A1 = M/dt + beta*K;
    B = M/dt - (1-beta)*K; % same as A for Crank-Nicholson
    % solve for t_i (stored in matrix T(t,i)
    with t timesteps and i nodes
    tp=zeros(floor(tmax/dt),1);
    tp(1)=0;
    for t = 1:floor(tmax/dt)
        tp(t+1)=tp(t)+dt;
        T(t+1,:) = A1\(B * T(t,:)' + F); %'
    end
else
    dt=.01;     % time step increment (s)
    tmax=10;      % final time (s)
    % set beta (parameter for weighting
    explicit and implicit calculation
    beta = .5;  % Crank-Nicholson: 0.5
    A1 = M/dt + beta*K;
    B = M/dt - (1-beta)*K; % same as A for Crank-Nicholson
    % solve for t_i (stored in matrix T(t,i)
    with t timesteps and i nodes
    tp=zeros(floor(tmax/dt),1);
    tp(1)=0;
    for t = 1:floor(tmax/dt)
        tp(t+1)=tp(t)+dt;
        T(t+1,:) = A1\(B * T(t,:)' + F); %'
    end
end
T;
%% compare to exact
if coord_type==1
    % exact (Abaqus FEA)
    exact=[0.              20.              20.
```

10.E-03	20.	138.526
20.E-03	20.	241.211
30.E-03	20.	331.23
40.E-03	20.	411.05
50.E-03	20.	482.598
60.E-03	20.	547.382
70.E-03	20.	606.594
80.E-03	20.	661.174
90.E-03	20.	711.873
100.E-03	20.	759.291
110.E-03	20.	803.91
120.E-03	20.	846.124
130.E-03	20.	886.252
140.E-03	20.	924.555
150.E-03	20.	961.253
160.E-03	20.	996.524
170.E-03	20.	1.03052E+03
180.E-03	20.	1.06337E+03
190.E-03	20.	1.09519E+03
200.E-03	20.	1.12606E+03
210.E-03	20.	1.15607E+03
220.E-03	20.	1.18528E+03
230.E-03	20.	1.21376E+03
240.E-03	20.	1.24157E+03
250.E-03	20.	1.26874E+03
260.E-03	20.	1.29532E+03
270.E-03	20.	1.32135E+03
280.E-03	20.	1.34687E+03
290.E-03	20.	1.37189E+03
300.E-03	20.	1.39646E+03
310.E-03	20.	1.42058E+03
320.E-03	20.	1.4443E+03
330.E-03	20.	1.46762E+03
340.E-03	20.	1.49057E+03
350.E-03	20.	1.51317E+03
360.E-03	20.	1.53543E+03
370.E-03	20.	1.55736E+03
380.E-03	20.	1.57898E+03
390.E-03	20.	1.6003E+03
400.E-03	20.	1.62134E+03
410.E-03	20.	1.6421E+03
420.E-03	20.	1.6626E+03
430.E-03	20.	1.68285E+03
440.E-03	20.	1.70285E+03
450.E-03	20.	1.72262E+03
460.E-03	20.	1.74216E+03
470.E-03	20.	1.76148E+03
480.E-03	20.	1.78058E+03
490.E-03	20.	1.79948E+03
500.E-03	20.	1.81818E+03
510.E-03	20.	1.83669E+03
520.E-03	20.	1.85501E+03
530.E-03	20.	1.87315E+03
540.E-03	20.	1.89111E+03
550.E-03	20.	1.9089E+03
560.E-03	20.	1.92653E+03
570.E-03	20.	1.94399E+03
580.E-03	20.	1.9613E+03
590.E-03	20.	1.97845E+03
600.E-03	20.	1.99545E+03
610.E-03	20.	2.01231E+03
620.E-03	20.	2.02903E+03
630.E-03	20.	2.0456E+03
640.E-03	20.	2.06205E+03
650.E-03	20.	2.07836E+03
660.E-03	20.	2.09454E+03
670.E-03	20.	2.1106E+03
680.E-03	20.	2.12654E+03

690.E-03	20.	2.14235E+03
700.E-03	20.	2.15805E+03
710.E-03	20.	2.17363E+03
720.E-03	20.	2.18911E+03
730.E-03	20.	2.20447E+03
740.E-03	20.	2.21972E+03
750.E-03	20.	2.23487E+03
760.E-03	20.	2.24992E+03
770.E-03	20.	2.26486E+03
780.E-03	20.	2.27971E+03
790.E-03	20.	2.29446E+03
800.E-03	20.	2.30911E+03
810.E-03	20.	2.32367E+03
820.E-03	20.	2.33814E+03
830.E-03	20.	2.35252E+03
840.E-03	20.	2.36681E+03
850.E-03	20.	2.38102E+03
860.E-03	20.	2.39514E+03
870.E-03	20.	2.40918E+03
880.E-03	20.	2.42313E+03
890.E-03	20.	2.437E+03
900.E-03	20.	2.4508E+03
910.E-03	20.	2.46451E+03
920.E-03	20.	2.47815E+03
930.E-03	20.	2.49172E+03
940.E-03	20.	2.50521E+03
950.E-03	20.	2.51863E+03
960.E-03	20.	2.53197E+03
970.E-03	20.	2.54525E+03
980.E-03	20.	2.55845E+03
990.E-03	20.	2.57159E+03
1.	20.	2.58466E+03
1.01	20.	2.59766E+03
1.02	20.	2.6106E+03
1.03	20.	2.62348E+03
1.04	20.	2.63629E+03
1.05	20.	2.64904E+03
1.06	20.	2.66172E+03
1.07	20.	2.67435E+03
1.08	20.	2.68692E+03
1.09	20.	2.69942E+03
1.1	20.	2.71187E+03
1.11	20.	2.72426E+03
1.12	20.	2.7366E+03
1.13	20.	2.74888E+03
1.14	20.	2.7611E+03
1.15	20.	2.77327E+03
1.16	20.	2.78539E+03
1.17	20.	2.79745E+03
1.18	20.	2.80946E+03
1.19	20.	2.82142E+03
1.2	20.	2.83333E+03
1.21	20.	2.84519E+03
1.22	20.	2.857E+03
1.23	20.	2.86876E+03
1.24	20.	2.88047E+03
1.25	20.	2.89214E+03
1.26	20.	2.90376E+03
1.27	20.	2.91533E+03
1.28	20.	2.92685E+03
1.29	20.	2.93833E+03
1.3	20.	2.94976E+03
1.31	20.	2.96115E+03
1.32	20.	2.9725E+03
1.33	20.	2.9838E+03
1.34	20.	2.99506E+03
1.35	20.	3.00627E+03
1.36	20.	3.01745E+03

```
    1.37            20.             3.02858E+03
    1.38            20.             3.03967E+03
    1.39            20.             3.05073E+03
    1.4             20.             3.06174E+03
    1.41            20.             3.07271E+03
    1.42            20.             3.08364E+03
    1.43            20.             3.09453E+03
    1.44            20.             3.10539E+03
    1.45            20.             3.1162E+03
    1.46            20.             3.12698E+03
    1.47            20.             3.13773E+03
    1.48            20.             3.14843E+03
    1.49            20.             3.1591E+03
    1.5             20.             3.16973E+03
    1.51            20.             3.18033E+03
    1.52            20.             3.19089E+03
    1.53            20.             3.20141E+03
    1.54            20.             3.21191E+03
    1.55            20.             3.22236E+03
    1.56            20.             3.23279E+03
    1.57            20.             3.24317E+03
    1.58            20.             3.25353E+03
    1.59            20.             3.26385E+03
    1.6             20.             3.27414E+03
    1.61            20.             3.2844E+03
    1.62            20.             3.29463E+03
    1.63            20.             3.30482E+03
    1.64            20.             3.31498E+03
    1.65            20.             3.32511E+03
    1.66            20.             3.33521E+03
    1.67            20.             3.34528E+03
    1.68            20.             3.35532E+03
    1.69            20.             3.36533E+03
    1.7             20.             3.37531E+03
    1.71            20.             3.38526E+03
    1.72            20.             3.39518E+03
    1.73            20.             3.40507E+03
    1.74            20.             3.41493E+03
    1.75            20.             3.42476E+03
    1.76            20.             3.43457E+03
    1.77            20.             3.44434E+03
    1.78            20.             3.45409E+03
    1.79            20.             3.46381E+03
    1.8             20.             3.47351E+03
    1.81            20.             3.48317E+03
    1.82            20.             3.49281E+03
    1.83            20.             3.50243E+03
    1.84            20.             3.51201E+03
    1.85            20.             3.52158E+03
    1.86            20.             3.53111E+03
    1.87            20.             3.54062E+03
    1.88            20.             3.5501E+03
    1.89            20.             3.55956E+03
    1.9             20.             3.56899E+03
    1.91            20.             3.5784E+03
    1.92            20.             3.58778E+03
    1.93            20.             3.59714E+03
    1.94            20.             3.60647E+03
    1.95            20.             3.61578E+03
    1.96            20.             3.62507E+03
    1.97            20.             3.63433E+03
    1.98            20.             3.64357E+03
    1.99            20.             3.65278E+03
    2.              20.             3.66197E+03];

t_exact=exact(:,1);
T1_exact=exact(:,2);
T2_exact=exact(:,3);
```

```
        h=figure;
        plot(tp,T(:,1),'.',t_exact,T2_exact,'-')
        xlabel('Time (s)')
        ylabel('Temperature (C)')
        legend('FEA (x=0)','Exact (x=0)','Location','NorthWest')
        axis([0 2 0 4000])
        %         h=figure;
        %         plot(tp,T(:,end),'.')
        %         xlabel('Time (s)')
        %         ylabel('Temperature (C)')
        %         text(2,500,['left/top final temp = ...
        ' num2str(T(end,1),'%s')])
        %         legend('FEA right/bottom edge',...
        'Location','NorthWest')
    elseif coord_type==2
        h=figure;
        % exact (Abaqus FEA)
        exact=[             0.                  20.                  20.
              500.E-03            25.2754              20.
                1.                27.357               20.
                1.5               28.9064              20.0002
                2.                30.1827              20.0017
                2.5               31.2857              20.0079
                3.                32.2661              20.023
                3.5               33.1541              20.0511
                4.                33.9694              20.0952
                4.5               34.7253              20.1571
                5.                35.432               20.2374
                5.5               36.0967              20.3363
                6.                36.7253              20.4532
                6.5               37.3224              20.5873
                7.                37.8918              20.7375
                7.5               38.4368              20.9027
                8.                38.9599              21.0819
                8.5               39.4635              21.2738
                9.                39.9496              21.4775
                9.5               40.4199              21.6919
               10.                40.8761              21.9162 ];

        t_exact=exact(:,1);
        T_exact_inner=exact(:,2);
        T_exact_outer=exact(:,3);
        plot(tp,T(:,1),'b.',t_exact,T_exact_inner,'b-', ...
            tp,T(:,end),'r.',t_exact,T_exact_outer,'r-')
        xlabel('Time (s)')
        ylabel('Temperature (C)')
        legend('FEA Inner Surface','Exact Inner Surface',...
        'FEA Outer Surface',...
        'Exact Outer Surface','Location','NorthWest')
%         axis([0 3600 10 110])
    else
        h=figure;
        % exact (Abaqus FEA)
        exact=[ 0.                  20.                  20.
              100.E-03            21.4997              20.
              200.E-03            22.5778              20.
              300.E-03            23.4033              20.
              400.E-03            24.0698              20.
              500.E-03            24.6311              20.
              600.E-03            25.1191              20.
              700.E-03            25.5537              20.
              800.E-03            25.9476              20.
              900.E-03            26.3093              20.
                1.                26.6448              20.
                1.1               26.9585              20.
                1.2               27.2536              20.0001
                1.3               27.5328              20.0001
```

1.4	27.7979	20.0002
1.5	28.0507	20.0003
1.6	28.2924	20.0005
1.7	28.5241	20.0007
1.8	28.7468	20.001
1.9	28.9613	20.0014
2.	29.1683	20.0019
2.1	29.3683	20.0026
2.2	29.562	20.0034
2.3	29.7497	20.0044
2.4	29.9318	20.0057
2.5	30.1088	20.0071
2.6	30.281	20.0088
2.7	30.4486	20.0108
2.8	30.6119	20.0131
2.9	30.7712	20.0156
3.	30.9267	20.0185
3.1	31.0786	20.0218
3.2	31.227	20.0254
3.3	31.3722	20.0294
3.4	31.5143	20.0339
3.5	31.6535	20.0387
3.6	31.7898	20.0439
3.7	31.9235	20.0496
3.8	32.0545	20.0557
3.9	32.1831	20.0623
4.	32.3093	20.0694
4.1	32.4333	20.0769
4.2	32.5551	20.0849
4.3	32.6748	20.0933
4.4	32.7924	20.1023
4.5	32.9081	20.1117
4.6	33.0219	20.1216
4.7	33.134	20.132
4.8	33.2442	20.1429
4.9	33.3528	20.1543
5.	33.4598	20.1661
5.1	33.5651	20.1785
5.2	33.669	20.1913
5.3	33.7713	20.2046
5.4	33.8722	20.2184
5.5	33.9717	20.2326
5.6	34.0699	20.2474
5.7	34.1668	20.2626
5.8	34.2623	20.2782
5.9	34.3567	20.2943
6.	34.4498	20.3109
6.1	34.5418	20.3279
6.2	34.6326	20.3454
6.3	34.7224	20.3633
6.4	34.811	20.3816
6.5	34.8986	20.4004
6.6	34.9852	20.4195
6.7	35.0708	20.4391
6.8	35.1554	20.4591
6.9	35.2391	20.4795
7.	35.3218	20.5003
7.1	35.4037	20.5215
7.2	35.4846	20.5431
7.3	35.5648	20.565
7.4	35.644	20.5873
7.5	35.7225	20.61
7.6	35.8002	20.6331
7.7	35.8771	20.6565
7.8	35.9532	20.6802
7.9	36.0286	20.7043
8.	36.1033	20.7287
8.1	36.1773	20.7535

```
                8.2                36.2506            20.7785
                8.3                36.3232            20.8039
                8.4                36.3951            20.8296
                8.5                36.4665            20.8557
                8.6                36.5371            20.882
                8.7                36.6072            20.9086
                8.8                36.6767            20.9355
                8.9                36.7455            20.9627
                9.                 36.8138            20.9902
                9.1                36.8816            21.0179
                9.2                36.9487            21.0459
                9.3                37.0154            21.0742
                9.4                37.0815            21.1027
                9.5                37.1471            21.1315
                9.6                37.2122            21.1605
                9.7                37.2768            21.1898
                9.8                37.3409            21.2194
                9.9                37.4046            21.2491
               10.                 37.4677            21.2791   ];
        t_exact=exact(:,1);
        T_exact_inner=exact(:,2);
        T_exact_outer=exact(:,3);
        plot(tp,T(:,1),'b.',t_exact,T_exact_inner,'b-', ...
            tp,T(:,end),'r.',t_exact,T_exact_outer,'r-')
        xlabel('Time (s)')
        ylabel('Temperature (C)')
        legend('FEA Inner Surface',...
        'Exact Inner Surface','FEA Outer Surface',...
        'Exact Outer Surface','Location','NorthWest')
        axis([0 10 18 38])
    end
    if coord_type==1
        filename='conduction_1D_heat_flux_cartesian';
        title('Cartesian Coordinates - Heat Flux BC''s')
    elseif coord_type==2
        filename='conduction_1D_heat_flux_cylindrical';
        title('Cylindrical Coordinates - Heat Flux BC''s')
    else
        filename='conduction_1D_heat_flux_spherical';
        title('Spherical Coordinates - Heat Flux BC''s')
    end
    saveas(h,filename,'png');
end
```

APPENDIX D

Matlab script for finite difference transient convection problem

```
clc; clear; %close all; format compact
%
h=40;
k=1;
c=800;
rho=2600;
alpha=k/(rho*c);
L=.08; % 8 cm
%
nnode=41;
dx=L/(nnode-1);
Bi=h*dx/k
Fo_max=1/(2*(1+Bi))
Fo=.4
%
% Fo=0.25;
% Bi=0.8;
%
Te=100;
dt=Fo*dx^2/alpha
%
t=0;
T=20*ones(1,nnode);
while t<3600
    t=t+dt;
    Tnew(1)=2*Fo*(T(2)+Bi*Te)+(1-2*Fo-2*Fo*Bi)*T(1);
    for i=2:nnode-1
        Tnew(i)=(1-2*Fo)*T(i)+Fo*(T(i-1)+T(i+1));
    end
    Tnew(nnode)=2*Fo*T(nnode-1)+(1-2*Fo)*T(nnode);
    T=Tnew;
    disp(['t_min, T = ' num2str(t/60,3) ' ' num2str(T,4)])
end
%
x=linspace(0,L,nnode);
figure
plot(x,T,'-')
xlabel('x')
ylabel('T')
%
% Abaqus FEA
%
fea=[    0.              76.2645
    2.E-03           74.3786
    4.E-03           72.5206
    6.E-03           70.6926
    8.E-03           68.8963
    10.E-03           67.1337
    12.E-03           65.4064
    14.E-03           63.7162
    16.E-03           62.0646
    18.E-03           60.4532
    20.E-03           58.8832
    22.E-03           57.3561
    24.E-03           55.8731
    26.E-03           54.4352
    28.E-03           53.0437
    30.E-03           51.6994
```

```
    32.E-03            50.4033
    34.E-03            49.1561
    36.E-03            47.9586
    38.E-03            46.8114
    40.E-03            45.7152
    42.E-03            44.6705
    44.E-03            43.6776
    46.E-03            42.7371
    48.E-03            41.8492
    50.E-03            41.0143
    52.E-03            40.2325
    54.E-03            39.5042
    56.E-03            38.8294
    58.E-03            38.2083
    60.E-03            37.641
    62.E-03            37.1276
    64.E-03            36.6681
    66.E-03            36.2627
    68.E-03            35.9113
    70.E-03            35.6139
    72.E-03            35.3706
    74.E-03            35.1813
    76.E-03            35.0461
    78.E-03            34.965
    80.E-03            34.938]    ;
x_fea=fea(:,1);
T_fea=fea(:,2);
hold on
plot(x_fea,T_fea,'r*')
legend('Finite Diff','Kaosi ist der Beste', 'Abaqus FEA')
title('Temperature distribution after 1 hour')
```

APPENDIX E

Matlab script for finite element transient convection problem

```
clc
clear all
format compact
close all

%% set constants

Tinf = 100; %deg C
h = 40; %W/m^2 K
T0 = 20; %deg C
rho = 2600; %kg/m^3
c = 800; %J/kg K
k = 1; %W/m K
% dx = 0.02; %m
L = 0.08; %m
Nnode=5; % 51;
dx=L/(Nnode-1)

%% calculate therm. diffus., Biot and Fourier nr.

% a = k / (rho * c); %thermal diffusivity
%
% Bi = h * dx / k %Biot number
% FoMax = 1 / (2 + 2*Bi) %Maximum Fourier number for stability
%
% Fo = 0.25; %set to .25 arbitrarily
% if Fo > FoMax
%     disp("Choose a lower Fourier number")
% end

%% determine time step

dt=60;

% dt = Fo * dx^2 / a;

%% calculate element stiffness matrix and assemble global stiff. mat.
A = 1; %area of slab similar to ex. 13.3
% in logans textbook for 1d problem

h*A*[1 0; 0 0]

K1 = A*k/dx*[1,-1;-1,1]+h*A*[1 0; 0 0]
K2 = A*k/dx*[1,-1;-1,1]
K = zeros(Nnode);
K(1:2,1:2)=K1;
for i = 2:Nnode-1
    K(i,i) = K(i,i)+ K2(1,1);
    K(i+1,i+1) = K(i+1,i+1) + K2(2,2);
    K(i,i+1) = K2(1,2);
    K(i+1,i) = K2(2,1);
end
K;

%% calculate element force vector and assemble global force vector

f = [0,0];

F = zeros(Nnode,1);

% for i = 1:4
```

```
%     F(i) = F(i) + f(1);
%     F(i+1) = F(i+1) +f(2);
% end

F(1) = A*h*Tinf;

F
%% calculate element mass capacitance matrix
% and assemble capac. matrix

% m = c*rho*A*L/6 *[2,1;1,2];
m = c*rho*A*dx/6 *[2,1;1,2];

M = zeros(Nnode);
for i = 1:Nnode-1
    M(i,i) = M(i,i)+ m(1,1);
    M(i+1,i+1) = M(i+1,i+1) + m(2,2);
    M(i,i+1) = m(1,2);
    M(i+1,i) = m(2,1);
end
M

%% calculate matrices for iteration

% set beta (parameter for weighting explicit and implicit calculation

beta = .5 % Crank-Nicholson: 0.5

A1 = M/dt + beta*K
B = M/dt - (1-beta)*K % same as A for Crank-Nicholson


% %solve for t_i (stored in matrix T(t,i)
with t timesteps and i nodes

% T(1,:) = A1\(B * T0*ones(Nnode,1) + F);

T(1,:) = T0*ones(Nnode,1);


tp=zeros(20,1);
tp(1)=0;
for t = 1:((70*60)/dt-1)
    tp(t+1)=tp(t)+dt;
    T(t+1,:) = A1\(B * T(t,:)' + F);
end

% T(1,:) = A1\B * T0*[1;1;1;1;1] + F;
%
% for t = 1:10
%     T(t+1,:) = A1\B * T(t,:)' + F;
% end

[tp T(:,1:5)]


exact=[0 20
    3.47  46.32
    6.93  53.30
    10.4  57.68
    13.87 60.85
    17.33 63.32
    20.80 65.33
    24.27 67.01
    27.72 68.45
    31.20 69.71
    34.67 70.82
    38.13 71.81
    41.60 72.72
    45.07 73.54
```

```
    48.53 74.31
    52.00 75.02
    55.47 75.69
    58.93 76.32
    62.40 76.92
    65.87 77.50
    69.33 78.05];

t_exact=60*exact(:,1);
T_exact=exact(:,2);

plot(tp,T(:,1),'.',t_exact,T_exact,'*')
xlabel('Time (s)')
ylabel('Temperature (C)')
legend('FEA','Exact','Location','NorthWest')
```

APPENDIX F

Matlab scripts for chapter 5 heat transfer problems

F.1 Matlab script for 1-dimensional finite element analysis of insulated rod cooling

```
clc
clear all
%% define const.

Tinf = 10; %deg C
T0 = 100; %deg C
h = 40; %W/m^2 K
Kxx = 0.195; %W/m K

%% calculate element stiffness matrix and assemble global stiff. mat.
A = pi/144; %area in m^2
L = 10/12; %length m
K1 = A*Kxx/L*[1,-1;-1,1];
K = zeros(5);
for i = 1:4
    K(i,i) = K(i,i)+ K1(1,1);
    K(i+1,i+1) = K(i+1,i+1) + K1(2,2);
    K(i,i+1) = K1(1,2);
    K(i+1,i) = K1(2,1);
end

K(5,5) = K(5,5) + h*A

%% calculate element force vector and assemble global force vector

f = [0,0];

F = zeros(5,1);
for i = 1:4
    F(i) = F(i) + f(1);
    F(i+1) = F(i+1) +f(2);
end

F(5) = A*h*Tinf;

%% solve system of equations K*t = F using boundary conditions
K(1,1) = 1;
F(1) = 100;
F(2) = -K(1,2) * F(1);
K(1,2) = 0;
K(2,1) = 0;

t = K \ F
```

F.2 Matlab script for 1-dimensional finite element analysis of rod cooling with free convection

```
clc
clear all
%% define const.

Tinf = 20; %deg F
T0 = 200; %deg F
h = 1; %Btu / (h ft^2 deg F)
Kxx = 3; %Btu / (h ft^2 deg F)
r = 2; %in
%% calculate element stiffness matrix and assemble global stiff. mat.
A = r^2*pi; %area in in^2
P = 2*pi*r; %perimeter in in
L = 3;
K1 = A*Kxx/L*[1,-1;-1,1] + h*P*L/6*[2,1;1,2];
K = zeros(4);
for i = 1:3
    K(i,i) = K(i,i)+ K1(1,1);
    K(i+1,i+1) = K(i+1,i+1) + K1(2,2);
    K(i,i+1) = K1(1,2);
    K(i+1,i) = K1(2,1);
end

K(4,4) = K(4,4) + h*A

%% calculate element force vector and assemble global force vector

f = h*Tinf*P*L/2*[1,1];

F = zeros(4,1);
for i = 1:3
    F(i) = F(i) + f(1);
    F(i+1) = F(i+1) +f(2);
end

F(4) = F(4) + h*Tinf*A;

%% solve system of equations K*t = F for t_2 to t_4
F(2) = 160*4*pi;
K(2:end,2:end) \ F(2:end)
```

F.3 Matlab script for 1-dimensional finite element analysis of transient fin

```
clc
clear all

%% set constants

Tinf = 25; %deg C
h = 150; %W/m^2 K
T0 = 25; %deg C
rho = 8900; %kg/m^3
c = 375; %J/kg K
k = 400; %W/m K
L = 0.02; %m
dx = L/2; %m
T1=85; % deg C

% Tinf = 100; %deg C
% h = 40; %W/m^2 K
% T0 = 20; %deg C
% rho = 2600; %kg/m^3
% c = 800; %J/kg K
```

```
% k = 1; %W/m K
% dx = 0.02; %m
% L = 0.08; %m

%% calculate therm. diffus., Biot and Fourier nr.

% a = k / (rho * c); %thermal diffusivity
%
% Bi = h * dx / k %Biot number
% FoMax = 1 / (2 + 2*Bi) %Maximum Fourier number for stability
%
% Fo = 0.25; %set to .25 arbitrarily
% if Fo > FoMax
%     disp("Choose a lower Fourier number")
% end

%% determine time step

% dt = Fo * dx^2 / a;

%% calculate element stiffness matrix and assemble global stiff. mat.
A = pi*.002^2; %area of slab similar to
% ex. 13.3 in logans textbook for 1d problem
P=2*pi*.002; % perimeter

K1 = A*k/dx*[1,-1;-1,1]+(h*P*dx/6)*[2 1; 1 2];
K = zeros(3);
for i = 1:2
    K(i,i) = K(i,i)+ K1(1,1);
    K(i+1,i+1) = K(i+1,i+1) + K1(2,2);
    K(i,i+1) = K1(1,2);
    K(i+1,i) = K1(2,1);
end
K

% A = 1; %area of slab similar to ex. 13.3
% in logans textbook for 1d problem
%
% K1 = A*k/dx*[1,-1;-1,1];
% K = zeros(5);
% for i = 1:4
%     K(i,i) = K(i,i)+ K1(1,1);
%     K(i+1,i+1) = K(i+1,i+1) + K1(2,2);
%     K(i,i+1) = K1(1,2);
%     K(i+1,i) = K1(2,1);
% end
% K

%% calculate element force vector and assemble global force vector

f=(h*Tinf*P*dx/2)*[1;1];
F = zeros(3,1);
for i = 1:2
    F(i) = F(i) + f(1);
    F(i+1) = F(i+1) +f(2);
end

F

% f = [0,0];
%
% F = zeros(5,1);
% for i = 1:4
%     F(i) = F(i) + f(1);
%     F(i+1) = F(i+1) +f(2);
% end
%
% F(1) = A*h*Tinf
```

```
%% calculate element mass capacitance
% matrix and assemble capac. matrix


m = c*rho*A*dx/6 *[2,1;1,2];
M = zeros(3);
for i = 1:2
    M(i,i) = M(i,i)+ m(1,1);
    M(i+1,i+1) = M(i+1,i+1) + m(2,2);
    M(i,i+1) = m(1,2);
    M(i+1,i) = m(2,1);
end
M

% m = c*rho*A*L/6 *[2,1;1,2];
% M = zeros(5);
% for i = 1:4
%     M(i,i) = M(i,i)+ m(1,1);
%     M(i+1,i+1) = M(i+1,i+1) + m(2,2);
%     M(i,i+1) = m(1,2);
%     M(i+1,i) = m(2,1);
% end
% M

%% calculate matrices for iteration

dt=0.1;


% set beta (parameter for weighting explicit and implicit calculation

beta=2/3;

A1 = M/dt + beta*K
B = M/dt - (1-beta)*K % same as A for Crank-Nicholson

A1p=A1(2:3,2:3)
Bp=B(2:3,2:3)
% %solve for t_i (stored in matrix T(t,i)
% with t timesteps and i nodes

A1p\Bp * T0*[1;1]
Fp=[F(2);F(3)]
[B(2,1);B(3,1)]*T1
[A1(2,1);A1(3,1)]*T1
T(1,:) = A1p\(Bp * T0*[1;1] + Fp +...
[B(2,1);B(3,1)]*T0 - [A1(2,1);A1(3,1)]*T1)
tp=zeros(20,1)
tp(1)=dt

for t = 1:29
    tp(t+1)=tp(t)+dt
    T(t+1,:) = A1p\(Bp * T(t,:)'... %'
    + Fp + [B(2,1);B(3,1)]*T1 - [A1(2,1);A1(3,1)]*T1);
end

[tp T]

plot(tp,T(:,1),'-',tp,T(:,2),'-')
xlabel('time')
ylabel('temperature')
legend('Node 2','Node 3')

% % set beta (parameter for weighting
% explicit and implicit calculation
%
% beta = 0.7; % Crank-Nicholson: 0.5
%
% A1 = M/dt + beta*K
```

```
% B = M/dt - (1-beta)*K % same as A for Crank-Nicholson
%
% % %solve for t_i (stored in matrix T(t,i)
%with t timesteps and i nodes
% T(1,:) = A1\B * T0*[1;1;1;1;1] + F;
%
% for t = 1:10
%     T(t+1,:) = A1\B * T(t,:)' + F;
% end
%
```

APPENDIX G

Matlab script for 1-dimensional transient finite element problem with phase change

```
function melting
% temperature BC (200 C) at x=0
clc; clear all; close all
%% set constants

T0 = 20; % deg C (initial condition)
T1 = 200; % deg C (left end BC)
k = 0.195; %W/m K
c = 2060; % = cp J/kg K
rho = 1200; %kg/m^3
hfs =3e6*rho/10; % latent heat of melting per deg C
%    (J/kg)*(kg/m^3)(1/s) = W/m^3

dt=.01;     % time step increment (s)
tmax=100;      % final time (s)

r = 1e-3; %radius of probe in m
A = pi*r^2; %area of top of pin
L = 6e-3; % height of probe in m
Nnode = 101; %
dx = L/(Nnode-1);
x=linspace(0,L,Nnode);

TL = 160;
TS = 150;

%% calculate element stiffness matrix and assemble global stiff. mat.

K2 = A*k/dx*[1,-1;-1,1];
K = zeros(Nnode);
% for i = 2:Nnode-1
for i = 1:Nnode-1
    K(i,i) = K(i,i)+ K2(1,1);
    K(i+1,i+1) = K(i+1,i+1) + K2(2,2);
    K(i,i+1) = K2(1,2);
    K(i+1,i) = K2(2,1);
end
K;

%% set beta (parameter for weighting explicit and implicit calculation

beta = .5;


%% calculate element mass capacitance matrix and assemble capac. matrix

m = c*rho*A*dx/6 *[2,1;1,2];
M = zeros(Nnode);
for i = 1:Nnode-1
    M(i,i) = M(i,i)+ m(1,1);
    M(i+1,i+1) = M(i+1,i+1) + m(2,2);
    M(i,i+1) = m(1,2);
    M(i+1,i) = m(2,1);
end
M;

%% calculate matrices for iteration

A1 = M/dt + beta*K;
```

```
% size(A1)
B = M/dt - (1-beta)*K; % same as A1 for Crank-Nicholson
% size(B)

%% iterate for each timestep while updating F

T(1,:) = T0*ones(Nnode,1);

tp=zeros(tmax/dt,1);
tp(1)=0;

for time_step=1:tmax/dt % t = dt:dt:tmax
    %     disp(['t, step = ' num2str([t,time_step])]);
    F = zeros(Nnode,1);
    f_type=zeros(Nnode-1,1);
    tp(time_step+1)=tp(time_step)+dt;

    for i = 1:Nnode-1
        [fnew,ftype] = fMelting(TL,TS,T(time_step,i),...
            T(time_step,i+1),dx,dt,A,hfs,rho);
        %         fnew=[0;0];  % for testing no melting
        f_type(i)=ftype;
        F(i) = F(i)+fnew(1);
        F(i+1) = F(i+1)+fnew(2);
    end
    % modify equations to impose BC at x=0
    Fred=zeros(Nnode-1,1);
    Fred(1)=Fred(1)+B(2,1)*T1-A1(2,1)*T1;
    Fred(2:Nnode-1)=F(3:Nnode);
    A1red=A1(2:Nnode,2:Nnode);
    Bred=B(2:Nnode,2:Nnode);
    Tred_prev=T(time_step,2:Nnode);

    Tred=A1red\(Bred * Tred_prev' + Fred);
    T(time_step+1,1) = T1;
    T(time_step+1,2:Nnode) = Tred;

    t_plot=tp(time_step+1);

    if abs(t_plot-10)<dt/10
        figure(1)
        t_plot;
        min_T=min(T(time_step+1,:));
        disp(['tplot, min_T= ' num2str([t_plot,min_T])])
        plot(x,T(time_step+1,:),'r*-')
        hold on
    elseif abs(t_plot-50)<dt/10
        figure(1)
        t_plot;
        min_T=min(T(time_step+1,:));
        disp(['tplot, min_T= ' num2str([t_plot,min_T])])
        plot(x,T(time_step+1,:),'g*-')
    elseif abs(t_plot-100)<dt/10
        figure(1)
        t_plot;
        min_T=min(T(time_step+1,:));
        disp(['tplot, min_T= ' num2str([t_plot,min_T])])
        plot(x,T(time_step+1,:),'b*-')
    end
end
hold on
plot_exact_soln

% Compaare to Abaqus solutions

function plot_exact_soln
%
```

```
% Abaqus case with melting (LH=3e6) - Table labels:
%    x, T(10 sec), T(50 sec), T(100 sec)
%
a=[  0.              200.              200.              200.
    60.E-06          184.634           192.964           194.972
    120.E-06          169.29            185.931           189.945
    180.E-06          153.99            178.903           184.919
    240.E-06          148.216           171.882           179.896
    300.E-06          142.517           164.871           174.877
    360.E-06          136.9             157.872           169.861
    420.E-06          131.376           153.013           164.851
    480.E-06          125.953           150.054           159.846
    540.E-06          120.639           147.548           155.809
    600.E-06          115.443           145.047           152.716
    660.E-06          110.37            142.553           150.404
    720.E-06          105.429           140.067           148.692
    780.E-06          100.627           137.592           146.985
    840.E-06           95.9677          135.129           145.283
    900.E-06           91.4576          132.681           143.586
    960.E-06           87.101           130.248           141.894
    1.02E-03         82.9018           127.832          140.21
    1.08E-03         78.8631           125.435          138.532
    1.14E-03         74.9871           123.058          136.861
    1.2E-03          71.2756           120.702          135.198
    1.26E-03         67.7295           118.368          133.543
    1.32E-03         64.349            116.057          131.896
    1.38E-03         61.1336           113.77           130.259
    1.44E-03         58.0821           111.508          128.632
    1.5E-03          55.1927           109.272          127.015
    1.56E-03         52.4629           107.062          125.408
    1.62E-03         49.8898           104.879          123.812
    1.68E-03         47.4698           102.724          122.228
    1.74E-03         45.199            100.597          120.656
    1.8E-03          43.073             98.499          119.096
    1.86E-03         41.087             96.4302         117.549
    1.92E-03         39.236             94.3912         116.015
    1.98E-03         37.5146            92.3825         114.495
    2.04E-03         35.9175            90.4045         112.99
    2.1E-03          34.4388            88.4576         111.499
    2.16E-03         33.0729            86.5423         110.022
    2.22E-03         31.814             84.6588         108.562
    2.28E-03         30.6562            82.8076         107.117
    2.34E-03         29.5939            80.9888         105.688
    2.4E-03          28.6211            79.2029         104.276
    2.46E-03         27.7324            77.4499         102.881
    2.52E-03         26.9223            75.7302         101.503
    2.58E-03         26.1854            74.0439         100.142
    2.64E-03         25.5165            72.3912          98.7999
    2.7E-03          24.9107            70.7722          97.4758
    2.76E-03         24.3632            69.1871          96.1703
    2.82E-03         23.8694            67.6359          94.8837
    2.88E-03         23.4251            66.1186          93.6163
    2.94E-03         23.0261            64.6354          92.3684
    3.E-03           22.6685            63.1863          91.1404
    3.06E-03         22.3488            61.7711          89.9324
    3.12E-03         22.0636            60.3899          88.7449
    3.18E-03         21.8096            59.0426          87.5779
    3.24E-03         21.5839            57.7291          86.4318
    3.3E-03          21.3837            56.4494          85.3069
    3.36E-03         21.2066            55.2032          84.2033
    3.42E-03         21.0502            53.9905          83.1214
    3.48E-03         20.9124            52.811           82.0613
    3.54E-03         20.7912            51.6647          81.0233
    3.6E-03          20.6848            50.5512          80.0075
    3.66E-03         20.5917            49.4704          79.0143
    3.72E-03         20.5103            48.4221          78.0437
    3.78E-03         20.4392            47.4059          77.0961
    3.84E-03         20.3774            46.4218          76.1715
```

```
    3.9E-03          20.3237          45.4693          75.2703
    3.96E-03         20.2771          44.5482          74.3924
    4.02E-03         20.2368          43.6583          73.5382
    4.08E-03         20.202           42.7992          72.7078
    4.14E-03         20.172           41.9707          71.9014
    4.2E-03          20.1462          41.1725          71.119
    4.26E-03         20.124           40.4042          70.3609
    4.32E-03         20.105           39.6655          69.6273
    4.38E-03         20.0888          38.9562          68.9181
    4.44E-03         20.0749          38.2759          68.2336
    4.5E-03          20.0631          37.6244          67.574
    4.56E-03         20.0531          37.0012          66.9392
    4.62E-03         20.0445          36.4062          66.3295
    4.68E-03         20.0373          35.839           65.7449
    4.74E-03         20.0312          35.2993          65.1856
    4.8E-03          20.0261          34.7869          64.6517
    4.86E-03         20.0217          34.3014          64.1432
    4.92E-03         20.0181          33.8426          63.6602
    4.98E-03         20.015           33.4102          63.2029
    5.04E-03         20.0125          33.004           62.7713
    5.1E-03          20.0103          32.6238          62.3654
    5.16E-03         20.0085          32.2692          61.9855
    5.22E-03         20.007           31.94            61.6314
    5.28E-03         20.0058          31.6362          61.3033
    5.34E-03         20.0048          31.3574          61.0013
    5.4E-03          20.004           31.1034          60.7254
    5.46E-03         20.0033          30.8742          60.4756
    5.52E-03         20.0027          30.6695          60.2519
    5.58E-03         20.0022          30.4893          60.0545
    5.64E-03         20.0019          30.3333          59.8834
    5.7E-03          20.0016          30.2015          59.7385
    5.76E-03         20.0014          30.0938          59.62
    5.82E-03         20.0012          30.0101          59.5277
    5.88E-03         20.0011          29.9504          59.4618
    5.94E-03         20.001           29.9146          59.4223
    6.E-03           20.001           29.9026          59.4091
] ;

x=a(:,1);
t10=a(:,2);
t50=a(:,3);
t100=a(:,4);
plot(x,t10,'r-')
plot(x,t50,'g-')
plot(x,t100,'b-')
% plot(x,t1,x,t5,x,t10,x,t20,x,t30,x,t40,x,t50)
xlabel('Distance (m)')
ylabel('Temperature (C)')
legend('Matlab t = 10 s','Matlab t = 50 s','Matlab t = 100 s',...
    'Abaqus t = 10 s','Abaqus t = 50 s','Abaqus t = 100 s')
title('Temperature Profile, h_{fs}=3e6')
% axis([ 0 .006 0 6000])
```

APPENDIX H

Function to compute force vector for 1-D melting problem

```
function [f,type] = fMelting(TL,TS,T1,T2,dx,dt,A,hfs,rho)
% local force vector for melting phenomena

% Consider four cases when T1 is always greater than T2
% (dT/dx<0 everywhere)
%    Note: TL>TS
% - type 1: T1>=T2, T1 and T2 both <= TS (all solid)
% - type 2: T1>=T2, T1>TS, T1<TL and T2<TS (part solid, part melting)
% - type 3: T1>=T2, T1>TS, T1<TL, T2>TS, T2<TL
% (all melting)
% - type 4: T1>=T2, T1>TL and T2<TS (part liquid,
% part melting, part solid)
% - type 5: T1>=T2, T1>TL, T2>TS, T2<TL (part liquid, part melting)
% - type 6: T1>=T2, T1>TL and T2>TL (all liquid)
% - type 7: T2>T1, T1 and T2 both <= TS (all solid)
% - type 8: T2>T1, T2>TS, T2<TL and T1<TS (part solid, part melting)
% - type 9: T2>T1, T1>TS, T1<TL, T2>TS, T2<TL
% (all melting)
% - type 10: T2>T1, T2>TL and T1<TS (part liquid,
% part melting, part solid)
% - type 11: T2>T1, T2>TL, T1>TS, T1<TL (part liquid, part melting)
% - type 12: T2>T1, T1>TL and T2>TL (all liquid)
%
[T1 T2 TS TL];
if (T1 >= T2) && (T1 <= TS) && (T2 <= TS) % f=[[0;0]
    type=1;
    f=[0;0];
elseif (T1 > TS) && (T1 <= TL) && (T2 <= TS)
    type=2;
    f = [-(hfs*A*dx)*(T1 - TS)*(T1 - 2*T2 + TS)/(2*(T1 - T2)^2);
         -(hfs*A*dx)*(T1 - TS)^2/(2*(T1 - T2)^2)];
elseif(T1 >= T2) && (T1 > TS) && (T1 <= TL) &&...
(T2 > TS) && (T2 <= TL)
    type=3;
    f = -(hfs*A*dx)/2*[1;1];
elseif (T1 >= T2) && (T1>TL) && (T2<TS)
    type=4;
    f = -[(hfs*A*dx)*(TL - TS)*(TL - 2*T2 + TS)/(2*(T1 - T2)^2);
          (hfs*A*dx)*(TL - TS)*(TL - 2*T1 + TS)/(2*(T1 - T2)^2)];
elseif (T1 >= T2) && (T1 > TL) && (T2 > TS) && (T2 <= TL)
    type=5;
    f = -[(hfs*A*dx)*(T2 - TL)^2/(2*(T1 - T2)^2);
          (hfs*A*dx)*(1/2 - (T1 - TL)^2/(2*(T1 - T2)^2))];
elseif (T1 >= T2) && (T1>TL) && (T2>TL)
    type=6;
    f=[0;0];
elseif (T2 > T1) && (T1 <= TS) && (T2 <= TS)
    type=7;
    f=[0;0];
elseif (T2 > T1) && (T2 > TS) && (T2 <= TL) && (T1 <= TS)
    type=8;
    f = -[(hfs*A*dx)*(T2 - TS)^2/(2*(T1 - T2)^2); ...
          (hfs*A*dx)*(1/2 - (T1 - TS)^2/...
          (2*(T1 - T2)^2));
elseif (T2 > T1) && (T1 > TS) && (T1 <= TL) &&...
(T2 > TS) && (T2 <= TL)
    type=9;
```

```
    f= -(hfs*A*dx)/2*[1;1];
elseif (T2 > T1) && (T2 > TL) && (T1 <= TS)
    type=10;
    f = -[-(hfs*A*dx)*(TL - TS)*(TL - 2*T2 + TS)/(2*(T1 - T2)^2);
          (hfs*A*dx)*(TL - TS)*(TL - 2*T1 + TS)/(2*(T1 - T2)^2)];
elseif (T2 > T1) && (T2 > TL) && (T1 > TS) && (T1 <= TL)
    type=11;
    f = -[(hfs*A*dx)*(T1 - TL)*(T1 - 2*T2 + TL)/(2*(T1 - T2)^2);
          (hfs*A*dx)*(T1 - TL)^2/(2*(T1 - T2)^2)];
elseif (T2 > T1) && (T1 > TL) && (T2 > TL)
    type=12;
    f=[0;0];
end
```

APPENDIX I

Function to compute 1-dimensional phase change using variable specific heat

```
function melting
% 1-D conduction with melting
clc; clear all; close all; format compact
set(groot,'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
% set constants
T0 = 20; % deg C (initial condition)
k = 0.195; % W/m K
c = 2060; % J/kg K
rho = 1200; % kg/m^3
% melting parameters
TL = 160; % liquidus temperature (C)
TS = 150; % solidus temperature (C)
c_melting=3e6/(TL-TS); % J/kg K (latent heat / temperature range)
%
% mesh and time step parameters
%
% coarse: (OK for temperature distribution)
% Nnode=101; % number of nodes;
% nstep=1000; % number of time steps
%
% fine: (better for melt pool growth
Nnode=1001; % number of nodes;
nstep=10000; % number of time steps
for coord_type=1:3 % (1- Cartesian, 2-Cylindrical, 3- Spherical)
    tic
    figure(coord_type)
    % calculate element stiffness matrix and assemble global stiff. mat.
    if coord_type==1
        T1 = 200; % deg C (left end BC)
        % time step parameters
        tmax=100; % final time (s)
        dt=tmax/nstep; % time step increment (s)
        beta=.5; % set beta (for weighting explicit vs. implicit)
        % geometric parameters
        r = 1e-3; % radius (m)
        A = pi*r^2; % area (m)
        L = 6e-3; % length (m)
        % define mesh
        x1=0; % left node coordinate (m)
        x2=L; % right node coordinate (m)
        dx=L/(Nnode-1);
        x=x1:dx:x2;
        Ke = A*k/dx*[1,-1;-1,1];
        K = zeros(Nnode);
        for i = 1:Nnode-1
            K(i,i) = K(i,i)+ Ke(1,1);
            K(i+1,i+1) = K(i+1,i+1) + Ke(2,2);
            K(i,i+1) = Ke(1,2);
            K(i+1,i) = Ke(2,1);
        end
    elseif coord_type==2
        T1 = 200; % deg C (left end BC)
        % time step parameters
```

```
    tmax=100; % final time (s)
    dt=tmax/nstep; % time step increment (s)
    beta=.5; % set beta (for weighting explicit vs. implicit)
    % geometric parameters
    L = 1e-2; % length (m)
    % define mesh
    x1=L/2; % left node coordinate (m)
    x2=L; % right node coordinate (m)
    dx=(x2-x1)/(Nnode-1);
    x=x1:dx:x2;
    K = zeros(Nnode);
    for i = 1:Nnode-1
        Ke = 2*pi*k*(x(i)+x(i+1))/(2*dx)*[1,-1;-1,1];
        K(i,i) = K(i,i)+ Ke(1,1);
        K(i+1,i+1) = K(i+1,i+1) + Ke(2,2);
        K(i,i+1) = Ke(1,2);
        K(i+1,i) = Ke(2,1);
    end
else
    T1 = 200; % deg C (left end BC)
    % time step parameters
    tmax=100; % final time (s)
    dt=tmax/nstep; % time step increment (s)
    beta=.5; % set beta (for weighting explicit vs. implicit)
    % geometric parameters
    L = 1e-2; % length (m)
    % define mesh
    x1=L/2; % left node coordinate (m)
    x2=L; % right node coordinate (m)
    dx=(x2-x1)/(Nnode-1);
    x=x1:dx:x2;
    K = zeros(Nnode);
    for i = 1:Nnode-1
        Ke=4*pi*k*(x(i)^2+x(i)*x(i+1)+x(i+1)^2)/ ...
            (3*dx)*[1,-1;-1,1];
        K(i,i) = K(i,i)+ Ke(1,1);
        K(i+1,i+1) = K(i+1,i+1) + Ke(2,2);
        K(i,i+1) = Ke(1,2);
        K(i+1,i) = Ke(2,1);
    end
end
% iterate for each timestep while adjusting heat capacity
clear T
T=zeros(nstep,Nnode);
T(1,:) = T0*ones(1,Nnode);
tp=zeros(nstep,1);
tp(1)=0;
for time_step=1:tmax/dt % t = dt:dt:tmax
    F = zeros(Nnode,1);
    tp(time_step+1)=tp(time_step)+dt;
    % compute M matrix based on previous step temperature
    M = zeros(Nnode);
    A1 = zeros(Nnode);
    B = zeros(Nnode);
    for i = 1:Nnode-1
        T_previous=(T(time_step,i+1)+T(time_step,i))/2;
        if T_previous<=TS || T_previous>TL
            c_element=c;
        else
            c_element=c_melting;
        end
        if coord_type==1
            m = c_element*rho*A*dx/6 *[2,1;1,2];
        elseif coord_type==2
            r1=x(i);
```

```
            r2=x(i+1);
            m(1,1) = 2*pi*c_element*rho*(r2-r1)*(r1/4+r2/12);
            m(1,2) = 2*pi*c_element*rho*(r2-r1)*(r1/12+r2/12);
            m(2,1) = m(1,2);
            m(2,2) = 2*pi*c_element*rho*(r2-r1)*(r1/12+r2/4);
        else
            r1=x(i);
            r2=x(i+1);
            m(1,1) = 4*pi*c_element*rho*(r2-r1)*(r1^2/5+r1*r2/10+r2^2/30);
            m(1,2) = 4*pi*c_element*rho*(-3*r1^5+5*r1^4*r2-5*r1*r2^4+...
                3*r2^5)/(60*(r2-r1)^2);
            m(2,1) = m(1,2);
            m(2,2) = 4*pi*c_element*rho*(r2-r1)*(r1^2/30+r1*r2/10+r2^2/5);
        end
        M(i,i) = M(i,i)+ m(1,1);
        M(i+1,i+1) = M(i+1,i+1) + m(2,2);
        M(i,i+1) = m(1,2);
        M(i+1,i) = m(2,1);
    end
    % compute A1 and B matrices
    A1 = M/dt + beta*K;
    B = M/dt - (1-beta)*K; % same as A1 for Crank-Nicholson
    % compute force vector
    Fred=zeros(Nnode-1,1);
    Fred(1)=Fred(1)+B(2,1)*T1-A1(2,1)*T1;
    Fred(2:Nnode-1)=F(3:Nnode);
    A1red=A1(2:Nnode,2:Nnode);
    Bred=B(2:Nnode,2:Nnode);
    Tred_prev=T(time_step,2:Nnode);
    Tred=A1red\(Bred * Tred_prev' + Fred);
    T(time_step+1,1) = T1;
    T(time_step+1,2:Nnode) = Tred;
    t_plot=tp(time_step+1);
    if abs(t_plot-tmax/10)<dt/10
        t_plot;
        min_T=min(T(time_step+1,:));
        disp(['tplot, min_T= ' num2str([t_plot,min_T])])
        plot(x,T(time_step+1,:),'r*')
        hold on
    elseif abs(t_plot-tmax/2)<dt/10
        t_plot;
        min_T=min(T(time_step+1,:));
        disp(['tplot, min_T= ' num2str([t_plot,min_T])])
        plot(x,T(time_step+1,:),'g*')
        hold on
    elseif abs(t_plot-tmax)<dt/10
        t_plot;
        min_T=min(T(time_step+1,:));
        disp(['tplot, min_T= ' num2str([t_plot,min_T])])
        plot(x,T(time_step+1,:),'b*')
        hold on
    end
end
plot_exact_soln(coord_type,x1)
if coord_type==1
    legend('Matlab t = 10 s', 'Matlab t = 50 s', ...
        'Matlab t = 100 s','Abaqus t = 10 s', ...
        'Abaqus t = 50 s', 'Abaqus t = 100 s')
    title('Cartesian Coordinates')
elseif coord_type==2
    legend('Matlab t = 10 s', 'Matlab t = 50 s', ...
        'Matlab t = 100 s','Abaqus t = 10 s', ...
        'Abaqus t = 50 s', 'Abaqus t = 100 s')
    title('Cylindrical Coordinates')
else
```

```
        legend('Matlab t = 10 s', 'Matlab t = 50 s','Matlab t = 100 s')
        % legend('Matlab t = 10 s', 'Matlab t = 50 s', ...
        % 'Matlab t = 100 s','Abaqus t = 10 s', ...
        % 'Abaqus t = 50 s', 'Abaqus t = 100 s')
        title('Spherical Coordinates')
    end
    set(groot,'defaulttextinterpreter','latex');
    set(groot, 'defaultAxesTickLabelInterpreter','latex');
    set(groot, 'defaultLegendInterpreter','latex');
    xlabel('Distance (m)')
    ylabel('Temperature ($^ncirc$ C)')
    %
    % plot melt front
    figure(4)
    xm=melt_front(nstep,TS,TL,x,tp,T);
    tm=tp(1:nstep);
    if coord_type==1
        plot(tm,xm,'r-')
    elseif coord_type==2
        plot(tm,xm,'g-')
    else
        plot(tm,xm,'b-')
    end
    hold on

    toc
end
xlabel('time (s)')
ylabel('melt front (m)')
title('Melt Pool Growth')
legend('Cartesian','Cylindrical','Spherical','Location','Northwest')
%
% ------------------------------------------------------------------
%
function xm=melt_front(nstep,TS,TL,x,tp,T)
%
% tm=zeros(1,nstep);
% xm=zeros(nstep,1);
for i=1:nstep
    %     f=abs((TL+TS)/2-T(i,:));
    %     [fmin,imin]=min(f);
    %     xm(i)=x(imin)-x(1);
    %
    temps=T(i,:);
    if max(temps)<(TL+TS)/2
        xm(i)=0;
    elseif min(temps)>=(TL+TS)/2
        xm(i)=x(end);
    else
        clear temps2 x2
        %     i1=find(temps>=(((TL+TS)/2)-4.9)&temps<=(((TL+TS)/2)+4.9));
        i1=find(temps>=(TS+.1)&temps<=(TL-.1));
        i1_size=size(i1);
        if i1_size(2)==0
            [min_dT,imin]=min(abs(temps-(TS+TL)/2));
            xm(i)=x(imin)-x(1);
            continue
        elseif i1_size(2)==1
            xm(i)=x(i1)-x(1);
            continue
        end
        temps2=temps(i1);
        x2=x(i1)-x(1);
        xm(i) =interp1(temps2,x2',(TL+TS)/2);
    end
end
```

```
%
% ------------------------------------------------------------------
%
function plot_exact_soln(coord_type,x1)
% Plot temperature vs. position
if coord_type==1
    %
    % Cartesian coordinates (LH=3e6) - Table labels:
    %    x, T(10 sec), T(50 sec), T(100 sec)
    %
    a=[  0.               200.              200.               200.
        60.E-06           184.634           192.964
194.972
        120.E-06           169.29            185.931
189.945
        180.E-06           153.99            178.903
184.919
        240.E-06           148.216           171.882
179.896
        300.E-06           142.517           164.871
174.877
        360.E-06           136.9             157.872
169.861
        420.E-06           131.376           153.013
164.851
        480.E-06           125.953           150.054
159.846
        540.E-06           120.639           147.548
155.809
        600.E-06           115.443           145.047
152.716
        660.E-06           110.37            142.553
150.404
        720.E-06           105.429           140.067
148.692
        780.E-06           100.627           137.592
146.985
        840.E-06            95.9677          135.129
145.283
        900.E-06            91.4576          132.681
143.586
        960.E-06            87.101           130.248
141.894
        1.02E-03          82.9018           127.832            140.21
        1.08E-03          78.8631           125.435
138.532
        1.14E-03          74.9871           123.058
136.861
        1.2E-03           71.2756           120.702
135.198
        1.26E-03          67.7295           118.368
133.543
        1.32E-03          64.349            116.057
131.896
        1.38E-03          61.1336           113.77
130.259
        1.44E-03          58.0821           111.508
128.632
        1.5E-03           55.1927           109.272
127.015
        1.56E-03          52.4629           107.062
125.408
        1.62E-03          49.8898           104.879
123.812
        1.68E-03          47.4698           102.724
122.228
        1.74E-03          45.199            100.597
```

120.656				
	1.8E-03	43.073	98.499	
119.096				
	1.86E-03	41.087	96.4302	
117.549				
	1.92E-03	39.236	94.3912	
116.015				
	1.98E-03	37.5146	92.3825	
114.495				
	2.04E-03	35.9175	90.4045	112.99
	2.1E-03	34.4388	88.4576	
111.499				
	2.16E-03	33.0729	86.5423	
110.022				
	2.22E-03	31.814	84.6588	
108.562				
	2.28E-03	30.6562	82.8076	
107.117				
	2.34E-03	29.5939	80.9888	
105.688				
	2.4E-03	28.6211	79.2029	
104.276				
	2.46E-03	27.7324	77.4499	
102.881				
	2.52E-03	26.9223	75.7302	
101.503				
	2.58E-03	26.1854	74.0439	
100.142				
	2.64E-03	25.5165	72.3912	
98.7999				
	2.7E-03	24.9107	70.7722	
97.4758				
	2.76E-03	24.3632	69.1871	
96.1703				
	2.82E-03	23.8694	67.6359	
94.8837				
	2.88E-03	23.4251	66.1186	
93.6163				
	2.94E-03	23.0261	64.6354	
92.3684				
	3.E-03	22.6685	63.1863	
91.1404				
	3.06E-03	22.3488	61.7711	
89.9324				
	3.12E-03	22.0636	60.3899	
88.7449				
	3.18E-03	21.8096	59.0426	
87.5779				
	3.24E-03	21.5839	57.7291	
86.4318				
	3.3E-03	21.3837	56.4494	
85.3069				
	3.36E-03	21.2066	55.2032	
84.2033				
	3.42E-03	21.0502	53.9905	
83.1214				
	3.48E-03	20.9124	52.811	
82.0613				
	3.54E-03	20.7912	51.6647	
81.0233				
	3.6E-03	20.6848	50.5512	
80.0075				
	3.66E-03	20.5917	49.4704	
79.0143				
	3.72E-03	20.5103	48.4221	
78.0437				
	3.78E-03	20.4392	47.4059	
77.0961				

	3.84E-03	20.3774	46.4218	
76.1715				
	3.9E-03	20.3237	45.4693	
75.2703				
	3.96E-03	20.2771	44.5482	
74.3924				
	4.02E-03	20.2368	43.6583	
73.5382				
	4.08E-03	20.202	42.7992	
72.7078				
	4.14E-03	20.172	41.9707	
71.9014				
	4.2E-03	20.1462	41.1725	
71.119				
	4.26E-03	20.124	40.4042	
70.3609				
	4.32E-03	20.105	39.6655	
69.6273				
	4.38E-03	20.0888	38.9562	
68.9181				
	4.44E-03	20.0749	38.2759	
68.2336				
	4.5E-03	20.0631	37.6244	
67.574				
	4.56E-03	20.0531	37.0012	
66.9392				
	4.62E-03	20.0445	36.4062	
66.3295				
	4.68E-03	20.0373	35.839	
65.7449				
	4.74E-03	20.0312	35.2993	
65.1856				
	4.8E-03	20.0261	34.7869	
64.6517				
	4.86E-03	20.0217	34.3014	
64.1432				
	4.92E-03	20.0181	33.8426	
63.6602				
	4.98E-03	20.015	33.4102	
63.2029				
	5.04E-03	20.0125	33.004	
62.7713				
	5.1E-03	20.0103	32.6238	
62.3654				
	5.16E-03	20.0085	32.2692	
61.9855				
	5.22E-03	20.007	31.94	
61.6314				
	5.28E-03	20.0058	31.6362	
61.3033				
	5.34E-03	20.0048	31.3574	
61.0013				
	5.4E-03	20.004	31.1034	
60.7254				
	5.46E-03	20.0033	30.8742	
60.4756				
	5.52E-03	20.0027	30.6695	
60.2519				
	5.58E-03	20.0022	30.4893	
60.0545				
	5.64E-03	20.0019	30.3333	
59.8834				
	5.7E-03	20.0016	30.2015	
59.7385				
	5.76E-03	20.0014	30.0938	59.62
	5.82E-03	20.0012	30.0101	
59.5277				
	5.88E-03	20.0011	29.9504	

```
59.4618
        5.94E-03          20.001              29.9146
59.4223
        6.E-03            20.001              29.9026
59.4091  ] ;
    %
    x=a(:,1);
    t10=a(:,2);
    t50=a(:,3);
    t100=a(:,4);
    plot(x+x1,t10,'r-')
    plot(x+x1,t50,'g-')
    plot(x+x1,t100,'b-')
elseif coord_type==2
    %
    % Cylindrical coordinates (LH=3e6) - Table labels:
    %    x, T(10 sec), T(50 sec), T(100 sec)
    %
    a=[         0.                200.
200.                200.
        49.9999E-06       185.98              193.63
195.307
        100.E-06           172.111             187.324
190.66
        150.E-06           158.404             181.084
186.06
        200.E-06           150.558             174.908
181.506
        250.E-06           145.385             168.798
176.998
        300.E-06           140.286             162.754
172.535
        350.E-06           135.267             156.775
168.117
        400.E-06           130.334             152.464
163.744
        450.E-06           125.492             149.766
159.415
        500.E-06           120.748             147.104
155.791
        550.E-06           116.106             144.478
152.854
        600.E-06           111.572             141.887
150.52
        650.E-06           107.15              139.331
148.719
        700.E-06           102.844             136.81
146.938
        750.E-06            98.6577            134.324
145.176
        800.E-06            94.5948            131.872
143.434
        850.E-06            90.6577            129.455
141.712
        900.E-06            86.8485            127.072
140.009
        950.E-06            83.169             124.723
138.327
        1.E-03            79.6203            122.408
136.664
        1.05E-03          76.2032            120.127
135.021
        1.1E-03           72.9179            117.88
133.398
        1.15E-03          69.7641            115.667
131.795
        1.2E-03           66.7412            113.487
```

130.212				
	1.25E-03	63.8482	111.341	
128.649				
	1.3E-03	61.0837	109.229	
127.107				
	1.35E-03	58.4461	107.15	
125.584				
	1.4E-03	55.9331	105.105	
124.082				
	1.45E-03	53.5426	103.092	
122.601				
	1.5E-03	51.2719	101.113	121.14
	1.55E-03	49.1183	99.1673	
119.699				
	1.6E-03	47.0787	97.2545	
118.279				
	1.65E-03	45.15	95.3748	
116.879				
	1.7E-03	43.3289	93.528	115.5
	1.75E-03	41.6118	91.7141	
114.142				
	1.8E-03	39.9954	89.933	
112.804				
	1.85E-03	38.476	88.1846	
111.488				
	1.9E-03	37.0498	86.4689	
110.192				
	1.95E-03	35.7131	84.7857	
108.917				
	2.E-03	34.4623	83.135	
107.663				
	2.05E-03	33.2935	81.5166	
106.431				
	2.1E-03	32.203	79.9305	
105.219				
	2.15E-03	31.1871	78.3765	
104.028				
	2.2E-03	30.2421	76.8546	
102.859				
	2.25E-03	29.3644	75.3645	
101.711				
	2.3E-03	28.5504	73.9063	
100.584				
	2.35E-03	27.7967	72.4797	
99.4777				
	2.4E-03	27.0998	71.0846	
98.3931				
	2.45E-03	26.4565	69.7209	
97.3298				
	2.5E-03	25.8634	68.3884	
96.2877				
	2.55E-03	25.3176	67.087	
95.267				
	2.6E-03	24.816	65.8165	
94.2675				
	2.65E-03	24.3558	64.5768	
93.2894				
	2.7E-03	23.9341	63.3677	
92.3326				
	2.75E-03	23.5484	62.1889	
91.3972				
	2.8E-03	23.1961	61.0404	
90.4831				
	2.85E-03	22.8748	59.922	
89.5903				
	2.9E-03	22.5822	58.8334	
88.7188				
	2.95E-03	22.3161	57.7745	

87.8687			
	3.E-03	22.0746	56.7451
87.0398			
	3.05E-03	21.8557	55.7449
86.2323			
	3.1E-03	21.6576	54.7739
85.446			
	3.15E-03	21.4785	53.8317
84.6811			
	3.2E-03	21.317	52.9183
83.9373			
	3.25E-03	21.1715	52.0333
83.2148			
	3.3E-03	21.0405	51.1765
82.5135			
	3.35E-03	20.923	50.3479
81.8334			
	3.4E-03	20.8175	49.5471
81.1744			
	3.45E-03	20.7231	48.7739
80.5365			
	3.5E-03	20.6387	48.0282
79.9197			
	3.55E-03	20.5633	47.3097
79.324			
	3.6E-03	20.4962	46.6182
78.7492			
	3.65E-03	20.4364	45.9536
78.1955			
	3.7E-03	20.3833	45.3155
77.6627			
	3.75E-03	20.3362	44.7038
77.1507			
	3.8E-03	20.2945	44.1183
76.6597			
	3.85E-03	20.2575	43.5588
76.1894			
	3.9E-03	20.225	43.0251
75.7398			
	3.95E-03	20.1962	42.5169
75.311			
	4.E-03	20.1709	42.0342
74.9028			
	4.05E-03	20.1487	41.5766
74.5152			
	4.1E-03	20.1292	41.144
74.1481			
	4.15E-03	20.1122	40.7363
73.8015			
	4.2E-03	20.0973	40.3531
73.4753			
	4.25E-03	20.0843	39.9944
73.1694			
	4.3E-03	20.073	39.66
72.8839			
	4.35E-03	20.0631	39.3497
72.6185			
	4.4E-03	20.0547	39.0633
72.3733			
	4.45E-03	20.0473	38.8006
72.1482			
	4.5E-03	20.0411	38.5615
71.9431			
	4.55E-03	20.0357	38.3459
71.7579			
	4.6E-03	20.0312	38.1536
71.5925			
	4.65E-03	20.0273	37.9844

```
71.447
        4.7E-03           20.0242           37.8381
71.3211
        4.75E-03          20.0216           37.7148
71.2148
        4.8E-03           20.0196           37.6141
71.1281
        4.85E-03          20.018            37.536
71.0607
        4.9E-03           20.017            37.4804
71.0128
        4.95E-03          20.0163           37.4472
70.9841
        5.E-03            20.0161           37.4361
70.9745];
    x=a(:,1);
    t10=a(:,2);
    t50=a(:,3);
    t100=a(:,4);
    plot(x+x1,t10,'r-')
    plot(x+x1,t50,'g-')
    plot(x+x1,t100,'b-')
else
    %
    % Spherical coordinates (LH=3e6) - Table labels:
    %    x, T(10 sec), T(50 sec), T(100 sec)
    %
    a=[         0.              200.
200.              200.
        49.9999E-06       185.576           192.892
194.635
        100.E-06           171.448           185.925
189.376
        150.E-06           157.623           179.096
184.219
        200.E-06           150.16            172.402
179.164
        250.E-06           144.348           165.841
174.207
        300.E-06           138.664           159.411
169.346
        350.E-06           133.115           154.286
164.579
        400.E-06           127.706           150.294
159.904
        450.E-06           122.441           147.047
155.916
        500.E-06           117.325           143.864
152.645
        550.E-06           112.362           140.744
150.021
        600.E-06           107.555           137.686
147.604
        650.E-06           102.906           134.69
145.231
        700.E-06            98.4162          131.754
142.903
        750.E-06            94.0876          128.877
140.618
        800.E-06            89.9202          126.059
138.376
        850.E-06            85.9137          123.299
136.177
        900.E-06            82.0674          120.596
134.02
        950.E-06            78.38            117.949
131.905
```

	1.E-03	74.8497	115.358	
129.831				
	1.05E-03	71.4745	112.821	
127.797				
	1.1E-03	68.2518	110.339	
125.804				
	1.15E-03	65.179	107.909	123.85
	1.2E-03	62.2528	105.533	
121.936				
	1.25E-03	59.47	103.208	120.06
	1.3E-03	56.8272	100.935	
118.223				
	1.35E-03	54.3206	98.7117	
116.423				
	1.4E-03	51.9465	96.5386	114.66
	1.45E-03	49.7008	94.4146	
112.934				
	1.5E-03	47.5796	92.3392	
111.243				
	1.55E-03	45.5786	90.3115	
109.588				
	1.6E-03	43.6937	88.331	
107.969				
	1.65E-03	41.9206	86.3969	
106.383				
	1.7E-03	40.255	84.5086	
104.832				
	1.75E-03	38.6925	82.6654	
103.314				
	1.8E-03	37.2288	80.8667	
101.829				
	1.85E-03	35.8597	79.1118	
100.377				
	1.9E-03	34.5808	77.4	
98.9574				
	1.95E-03	33.388	75.7307	
97.5691				
	2.E-03	32.2769	74.1032	
96.2122				
	2.05E-03	31.2435	72.517	
94.8861				
	2.1E-03	30.2837	70.9713	
93.5905				
	2.15E-03	29.3937	69.4657	
92.3249				
	2.2E-03	28.5694	67.9993	
91.0891				
	2.25E-03	27.8073	66.5717	
89.8825				
	2.3E-03	27.1035	65.1822	
88.7049				
	2.35E-03	26.4547	63.8303	
87.5559				
	2.4E-03	25.8574	62.5153	
86.4351				
	2.45E-03	25.3082	61.2366	
85.3422				
	2.5E-03	24.8042	59.9938	
84.2768				
	2.55E-03	24.3421	58.7861	
83.2387				
	2.6E-03	23.9193	57.6132	
82.2274				
	2.65E-03	23.5329	56.4744	
81.2428				
	2.7E-03	23.1802	55.3691	
80.2844				
	2.75E-03	22.8589	54.2969	

79.352			
	2.8E-03	22.5666	53.2572
78.4452			
	2.85E-03	22.301	52.2495
77.5639			
	2.9E-03	22.0602	51.2733
76.7076			
	2.95E-03	21.842	50.3281
75.8762			
	3.E-03	21.6446	49.4133
75.0693			
	3.05E-03	21.4664	48.5286
74.2867			
	3.1E-03	21.3057	47.6734
73.5281			
	3.15E-03	21.161	46.8473
72.7932			
	3.2E-03	21.031	46.0497
72.0819			
	3.25E-03	20.9142	45.2803
71.3937			
	3.3E-03	20.8096	44.5386
70.7286			
	3.35E-03	20.7159	43.8242
70.0863			
	3.4E-03	20.6322	43.1365
69.4664			
	3.45E-03	20.5575	42.4753
68.8688			
	3.5E-03	20.4909	41.84
68.2933			
	3.55E-03	20.4317	41.2304
67.7396			
	3.6E-03	20.3791	40.6459
67.2074			
	3.65E-03	20.3325	40.0862
66.6967			
	3.7E-03	20.2912	39.5509
66.2071			
	3.75E-03	20.2546	39.0397
65.7384			
	3.8E-03	20.2224	38.5521
65.2905			
	3.85E-03	20.194	38.0878
64.863			
	3.9E-03	20.1689	37.6465
64.4559			
	3.95E-03	20.1469	37.2278
64.0689			
	4.E-03	20.1277	36.8314
63.7018			
	4.05E-03	20.1108	36.4569
63.3543			
	4.1E-03	20.096	36.1041
63.0264			
	4.15E-03	20.0831	35.7726
62.7178			
	4.2E-03	20.0718	35.4622
62.4283			
	4.25E-03	20.0621	35.1724
62.1578			
	4.3E-03	20.0536	34.9032
61.9059			
	4.35E-03	20.0463	34.6541
61.6727			
	4.4E-03	20.0399	34.4249
61.4578			
	4.45E-03	20.0345	34.2154

```
61.2611
        4.5E-03             20.0299             34.0252
61.0824
        4.55E-03            20.0259             33.8542
60.9215
        4.6E-03             20.0226             33.7021
60.7783
        4.65E-03            20.0198             33.5687
60.6526
        4.7E-03             20.0174             33.4537
60.5442
        4.75E-03            20.0156             33.357
60.4529
        4.8E-03             20.0141             33.2782
60.3786
        4.85E-03            20.013              33.2174
60.3211
        4.9E-03             20.0122             33.1741
60.2803
        4.95E-03            20.0117             33.1483
60.2559
        5.E-03              20.0116             33.1397
60.2478 ];
    x=a(:,1);
    t10=a(:,2);
    t50=a(:,3);
    t100=a(:,4);
    plot(x+x1,t10,'r-')
    plot(x+x1,t50,'g-')
    plot(x+x1,t100,'b-')
end
```

BIBLIOGRAPHY

[Online]. Available: https://abaqus-docs.mit.edu/2017/English/SIMACAEBMKRefMap/simabmk-c-freezingofsolid.htm

D., G. M., *Advanced Engineering Mathematics*. Prentice-Hall, 1998.

Holman, J., *Heat Transfer*. Mcgraw-Hill, 2010.

Lazaridis, A., "Numerical solution of the multidimensional solidification (or melting) problem," *International Journal of Heat and Mass Transfer*, vol. 13, pp. 1459–1477, 1970.

Logan, D. L., *A First Course in the Finite Element Method*. Brooks/Cole, 2012.

Mabrouk, R., Dhahri, H., Naji, H., Hammouda, S., and Younsi, Z., "Lattice boltzmann simulation of forced convection melting of a composite phase change material with heat dissipation through an open-ended channel," *International Journal of Heat and Mass Transfer*, vol. 153, p. 119606, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0017931019343157

Mills, A. F., *Heat Transfer*. R.R. Donnelly Sons, 1992.

Ogoh, W. and Groulx, D., "Stefan's problem: Validation of a one-dimensional solid-liquid phase change heat transfer process," in *Comsol Conference 2010*, 2010.

Sadd, M. H., *Elasticity: theory, applications, and numerics*. Academic Press, 2014.

Stephan, P., Schaber, K., Stephan, K., and Mayinger, F., *Thermodynamik: Grundlagen und technische Anwendungen Band 1: Einstoffsysteme*. Springer Berlin Heidelberg, 2013.

Yaakoubi, M., Kchaou, M., and Dammak, F., "Simulation of the thermomechanical and metallurgical behavior of steels by using abaqus software," *Computational Materials Science*, vol. 68, pp. 297–306, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927025612005824

Zhang, W.-J., Liu, Z.-Y., Liu, Z.-L., and Cai, L.-C., "Melting curves and entropy of melting of iron under earth's core conditions," *Physics of the Earth and Planetary Interiors*, vol. 244, pp. 69–77, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031920114002222