11-9-2010

Solution of Nonlinear Transient Heat Transfer Problems

Donovan O. Buckley
Florida International University, dbucks23@gmail.com

DOI: 10.25148/etd.FI10120804

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

SOLUTION OF NONLINEAR TRANSIENT HEAT TRANSFER PROBLEMS

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

MECHANICAL ENGINEERING

by

Donovan Buckley

2010

To: Dean Amir Mirmiran
College of Engineering and Computing

This thesis, written by Donovan Buckley, and entitled Solution of Nonlinear Transient Heat Transfer Problems, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

George Dulikravich

Yiding Cao

Igor Tsukanov, Major Professor

Date of Defense: November 9, 2010

The thesis of Donovan Buckley is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Interim Dean Kevin O'Shea
University Graduate School

Florida International University, 2010

ACKNOWLEDGMENTS

First, let me give GOD all the praise and accept the glory of having accomplished an educational goal, which I set for myself three years ago. I would like to take this opportunity to express my gratitude to my family and friends who were there for me as a source of strength and encouragement. I would like to especially thank my wife, Marsha, for her unwavering support of me pursuing this goal. I do not have enough words to express my appreciation for her dedication, love, and continued confidence in me. I owe her for being so unselfish, which has taken the load off my shoulder and made this experience much easier.

I am very grateful to my major professor Dr. Igor Tsukanov for his support, guidance and patience throughout my thesis project. I am still impressed by his comprehensive knowledge in mathematics, numerical computation and his analytical approach to problems. This inspired me, enriched my growth as a student and nourished my intellectual maturity, which I will benefit from throughout my engineering career. I am extremely appreciative of how accessible he was, whether it was late at nights or on weekends, he was always reachable to answer any questions I had.

I thank also my committee members Dr. George Dulikravich and Dr. Yiding Cao for their critical comments and suggestions they shared with me. I thank them for their valuable advice and guidance, and for all the journal articles they recommended that I used as references during my research.

ABSTRACT OF THE THESIS

SOLUTION OF NONLINEAR TRANSIENT HEAT TRANSFER PROBLEMS

by

Donovan Buckley

Florida International University, 2010

Miami, Florida

Professor Igor Tsukanov, Major Professor

In the presented thesis work, meshfree method with distance fields was extended to obtain solution of nonlinear transient heat transfer problems. The thesis work involved development and implementation of numerical algorithms, data structure, and software. Numerical and computational properties of the meshfree method with distance fields were investigated. Convergence and accuracy of the methodology was validated by analytical solutions, and solutions produced by commercial FEM software (ANSYS 12.1).

The research was focused on nonlinearities caused by temperature-dependent thermal conductivity. The behavior of the developed numerical algorithms was observed for both weak and strong temperature-dependency of thermal conductivity. Oseen and Newton-Kantorovich linearization techniques were applied to linearized the governing equation and boundary conditions. Results of the numerical experiments showed that the meshfree method with distance fields has the potential to produced fast accurate solutions. The method enables all prescribed boundary conditions to be satisfied exactly.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

CHAPTER 1
INTRODUCTION

1.1 Importance of Nonlinear Heat Transfer Solutions

Most metallic materials have thermal properties (thermal conductivity, specific heat, and density) that are usually temperature-dependent. This results in nonlinearities in the governing equations and the boundary conditions describing the temperature distribution through these materials. However, because of the difficulties associated with the solution of these nonlinear heat transfer problems, simplifying assumptions are usually made to linearize such problems. For example, in the case of materials that have thermal conductivity which varies slightly with temperature, constant thermal conductivity is generally assumed. However, if temperature change is substantial or the thermal conductivity varies greatly with temperature, the assumption of constant thermal conductivity may lead to significant error in the solution. Therefore, when modeling and simulating temperature distribution for such problems, nonlinearities caused by temperature-dependent thermal conductivity have to be accounted for by the numerical computation.

Some materials, for example PTFE and Indium have a very weak dependence of their thermal properties on the temperature. For such materials, thermal properties can be assumed to be constant. In contrast, materials such as, Alumina, Hastelloy C-2000 alloy, and Copper can have large variation in their thermal properties. Copper for example, has very steep temperature-dependency of thermal conductivity in range of very low temperatures [5]. The thermal conductivity of copper at $T = 1°$K is 5730 W/m^2-K and when $T = 40°$K thermal conductivity increases sharply to 19600 W/m^2-K. In this case, temperature-dependence of thermal conductivity cannot be neglected and has to be incorporated into the solution procedure.

Many practical engineering situations require solving nonlinear transient heat transfer problems. However, exact analytical solutions of such nonlinear transient heat transfer problems are generally not available. Due to the limitations of analytical solutions for nonlinear heat transfer problems, a number of numerical methods have been developed to solve such problems. In the presented thesis work, the meshfree method with distance fields is adopted to obtain numerical solutions of such nonlinear transient heat transfer problems.

1.2 Nonlinear Heat Transfer Problems

Temperature dependence of the material properties causes nonlinearity in the differential equation, nonlinearity in the boundary conditions or nonlinearities in both. Based on the mathematical formulation of the corresponding boundary value problem, nonlinear heat transfer problems can be classified into three groups [15]:

1. *Materials with temperature-dependent properties:* This would be the case of temperature-dependent thermal conductivity, density, and/or specific heat capacity. This gives rise to a nonlinear partial differential equation in the form

$$\rho(T)c(T)\frac{\partial T}{\partial t} = div[\lambda(T)\nabla T] + Q, \tag{1.1}$$

   where T is temperature, $\lambda(T)$ is the temperature-dependent thermal conductivity of the medium, $\rho(T)$ is temperature-dependent density, $c(T)$ is the temperature-dependent specific heat capacity and Q is the internal heat generation. For the problems that we investigated in this thesis work, density and specific heat capacity are assumed to be constant and there is no internal heat generation.

2. *Nonlinear boundary conditions:* These are caused by heat radiation or temperature-dependent heat transfer coefficients. For this thesis, the focused was on investigating numerical approaches for solution of problems with nonlinear boundary conditions of the following kinds:

   - Prescribed heat flux or Neumann boundary condition

$$q = \bar{q} \qquad where \qquad q = \lambda(T)\frac{\partial T}{\partial n} \tag{1.2}$$

   - Convective boundary condition

$$q = -\alpha(T - T_{env}), \tag{1.3}$$

     in which α is the convective heat coefficient and T_{env} is the temperature of the medium surrounding the convective boundary. The nonlinear convective boundary condition can be written as

$$\lambda(T)\frac{\partial T}{\partial n} = -\alpha(T - T_{env}) \tag{1.4}$$

$$\frac{\partial T}{\partial n} + \frac{\alpha}{\lambda(T)}T = \frac{\alpha}{\lambda(T)}T_{env} \tag{1.5}$$

- Radiation boundary condition

$$q = \sigma\varepsilon(T^4 - T_r^4), \tag{1.6}$$

where σ is the Stefan-Boltzmann constant and ε in the emissivity between the surface and the boundary at temperature T_r.

3. *Nonlinear sources:* These are characteristic of some kind of chemical reaction or phase transition taking place within the solid medium.

1.3 Computation Techniques

A variety of engineering analysis methods are available for solution of nonlinear transient heat transfer problems. A commonly used engineering analysis method for solution of these problems is the finite-difference method (FDM) [18]. Finite element method (FEM) [2] is also a commonly used method to solve these problems. The primary advantages of the finite element method over the finite difference method are that irregular boundaries can be handled easily, and the size of the finite element can be varied easily over the region. Another method used is the boundary element method (BEM) [26], where the numerical solution of the continuum is performed with a reduction of dimensionality of the problem. The success of BEM is that the number of the resulting simultaneous equations depends only upon the discretization of the boundary of the domain and that technique can be employed to represent the solution over the boundary elements. Thus the problem can be treated with one less dimension.

Out of all the engineering analysis methods mentioned above and others that have been developed so far, the FEM has found to be the most used. However, FEM relies on various spacial discretizations (meshes, grids, etc.) that have to *conform* to the shape of the geometric object. Creation of such spatial discretization known as meshing, can be a difficult and computationally expensive procedure. These traditional methods of engineering analysis methods (FEM, BEM, etc), rely on the spatial discretization (meshing) of the geometric domain and/or its boundary to enforce or approximate the imposed boundary conditions at discrete location.

Although there has been tremendous advances in meshing technology that allows for automatic meshing of most geometries, the appropriate meshing remain challenging for complicated geometries where meshing can dominate manual and computer solution time. Also, once meshes are constructed, they severely constrain the geometric model, thus limiting possible changes to the geometry, motions, and deformations such as those needed for shape optimization and dynamic simulations [24]. This stimulated the development of alternative engineering analysis methods known as meshfree methods, which employ spatial discretizations that do not necessarily have to conform to the shape of the geometric model.

By contrast, meshfree methods discretize not the geometric domain but the underlying functional space. In meshfree systems, the geometric model of the domain neither conforms to nor is restricted by spacial discretization. Meshfree systems for engineering analysis therefore offer numerous advantages such as better handling of moving boundaries and changing geometry, over systems that are based on the traditional mesh-based methods. But they also require radical approaches to enforcing boundary conditions. Therefore, one of the main challenges for meshfree methods lies in constructing solutions to boundary value problems that satisfies the boundary conditions. In recent years a number of techniques with basis functions that do not have to conform to the geometry of the domain have been developed, example: Smooth Particle Hydrodynamics (SPH) [11], the diffuse element method (DEM) [14], the HP cloud method [3], partition of unity methods (PUM) [13], the reproducing kernel particle method (RKPM) [9], and others.

The geometric non-conformance of all such meshfree methods makes treatment of the boundary condition more challenging. Proposed remedies for this problem include the combination of Element Free Galerkin Method (EFG) with finite element shape functions near the boundary [7], window or correction functions that vanishes on the boundary [3], the use of modified variational principle [10], and lagrange multipliers. Although these techniques appeared promising, they often contradict the meaning of meshfree because of the nature of the approximation near the boundary.

In this thesis work, we adopted the meshfree method with distance fields, also called R-function method (RFM), to support solutions of nonlinear transient heat transfer problems. The salient feature of this method is that it allows all prescribed boundary to be satisfied exactly on all boundary points.

1.4 Meshfree Method with Distance Fields

1.4.1 Basic idea of meshfree method with distance fields

In this thesis work, we employed meshfree methods with distance fields to support solutions of nonlinear transient heat transfer problems. The initial idea of the meshfree method with distance fields was first proposed by Kantorovich [6]. Kantorovich proposed to represent a field that satisfied homogeneous Dirichlet boundary conditions as a product of two functions: $u = \omega\Phi$, where

1. ω is an implicit representation of geometry with zero set corresponding to the geometry and non-vanishing gradient at all points of the zero set [6].
2. Φ is an unknown function that allows to satisfy (exactly or approximately) the differential equation of the problem.

A key feature of this formulation is the ability to exactly satisfy boundary conditions on the zero set of ω. The idea at first appeared to have limited use because it was not clear at the time how to construct function ω for complex shapes, and because the method did not seem to generalized to other types of boundary value problems.

Several years later, Rvachev proposed that functions taking on zero value on boundary of the geometric domain can be constructed for virtually any geometric object using the theory of R-functions [23]. R-functions serves as a construction toolkit transforming a set-theoretic description of the boundary of the geometric object into a real valued function whose zero set coincides with the boundary. Details on R-functions and implementation techniques can be found in [1, 23]. Functions constructed using R-functions behave as distance to the boundaries near the boundary points and posses desired differential properties required for solution of the boundary value problem. In meshfree method with distance fields we are using the theory of R-functions to construct the approximate distance fields, therefore, meshfree method with distance fields is often referred to as R-function method. However, besides techniques based on the theory of R-functions, other methods may also be applied for construction of approximate distance fields. For example, the level set method [19], which results in distance-like functions can be used [4].

Representing boundaries of a geometric object by approximate distance fields made possible the extension of Kantorovich initial idea into the meshfree method with distance fields. Shapiro and Tsukanov showed that the method may be completely automated for a wide class of geometric and physical problems in a common meshfree environment [24]. Theoretical completeness of the method is shown in [17]. Meshfree method with distance fields allows the satisfaction of many types of boundary conditions exactly by employing solution structures that incorporate boundary conditions, approximate distance fields, and basis functions with unknown coefficients [17].

1.4.2 Principle of meshfree method with distance fields

The idea of the method is based on the observation that the solution of a differential equation with Dirichlet boundary conditions

$$u|_{\partial\Omega} = \varphi, \tag{1.7}$$

can be represented in the form

$$u = \omega\Phi + \varphi, \tag{1.8}$$

where ω is a distance field to the boundary $\partial\Omega$, and Φ is an arbitrary function. The distance field ω is constructed to take on zero value on the boundary $\partial\Omega$ and is positive in the interior of the domain Ω (Figure 1.1). Function u satisfies the prescribed Dirichlet boundary condition regardless of the function Φ. This representation of the solution in expression (1.8) is classified as a solution structure. The advantages of representing the solution u as a solution structure of this form are:

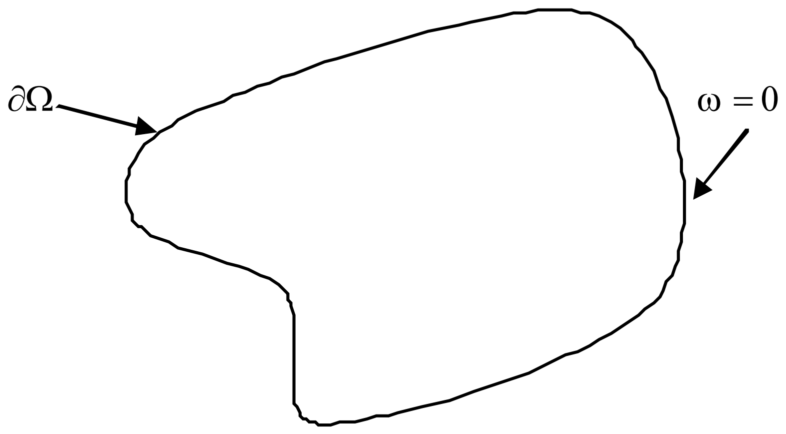


Figure 1.1: The domain for the heat transfer problem

- The expression (1.8) includes two independent types of information:
    1. The function ω completely describes all the geometric information of the problem.
    2. The function Φ whose sole purpose is to satisfy the analytical constraints of the boundary value problem exactly or approximately.
- Since ω is constructed to be identically zero on the boundary $\partial\Omega$ of the geometric domain, any function u of the form expression (1.8) will satisfy the boundary condition (1.7) exactly, independently of the properties of the unknown function Φ or the type of differential equation.
- Expression (1.8) contains no information about the differential equation of the boundary value problem. Rather, it represents the structure of the given geometric constraints.
- For any given boundary value problem, determination of the unknown Φ translates into solution to the boundary value problem. Since we cannot expect to determine such Φ exactly, we can approximate it by a finite-independent series

$$\Phi = \sum_{i=1}^{n} C_i \chi_i \tag{1.9}$$

  where C_i are scalar coefficient and χ_i are some basis function.
- The solution structure (1.8) does not place any constraint on the choice of the functions χ_i that approximate the function Φ. And in particular, the choice of the

coordinate functions does not depend on any particular spatial discretization of the geometric domains or its boundary conditions.

- For any given boundary value problem and a choice of the coordinate basis χ_i, the approximate solution is obtained as

$$u = \omega \sum_{i=0}^{n} C_i \chi_i \tag{1.10}$$

  and a variety of numerical methods can be used to solve for the numerical values of the coefficients C_i.

From a computational point of view, the intrinsic advantage of this procedure is in the clean modular separation of the geometric information represented by the function ω from the differential equation and the numerical method used to determine the unknown function Φ.

1.4.3 Solution Structures for Any Boundary Conditions

Rvachev, later noticed that the expression (1.8) is a zero order Taylor series expansion of u by the powers of the distance field ω with the product $\omega\Phi$ playing the role of a remainder term. Generalizing Kantorovich idea, Rvachev showed that any function can be represented by the powers of a distance field ω [16]:

$$u = u_0^* + \sum_{i=1}^{m} u_i^* \frac{\omega^i}{i!} + \omega^{m+1}\Phi \tag{1.11}$$

This power series looks very familiar to the classical Taylor series. In fact, if coefficients $u_{i\,i=0}^{*m}$ represent normal derivatives of u prescribed at the zero set of the distance field ω, the power series (1.11) represent a generalized Taylor series expansion of the function u by the powers of the distance field ω.

This straightforward generalization of Kantorovich idea allows systematic construction of solution structures for any and all boundary conditions. In each case, the solution structure will exactly interpolate all values and derivatives prescribed on the boundary and will contain necessary degrees of freedom (approximating the remainder term in the Taylor series expansion) in order to approximate the governing equations of the problem. It can be shown that such a solution structure forms a complete space of functions that satisfy the given boundary conditions exactly and approximate the governing equation of the problem [17].

However in order to represent normal derivatives of u prescribed on the boundary $\partial\Omega$, these coefficients $u^{*m}_{i\,i=0}$ have to be constant up to mth order of the normal direction to the boundary. This implies that

$$\frac{\partial^k u_i^*}{\partial n^k}|_{\partial\Omega} = 0, \quad i = 1, ..., m. \tag{1.12}$$

Most functions prescribed as boundary conditions do not satisfy condition (1.12) automatically. An operation known as conditioning of the function is used to transform any function u into a function satisfying the condition (1.12). One satisfactory means to condition a function appears as follows:

$$u^* = u - \sum_{i=0}^{m} \frac{1}{i!}\omega^i D_i^{\omega}(u), \tag{1.13}$$

where the differential operator D_i^{ω} is given as

$$D_i^{\omega}() = (\nabla() \cdot \nabla\omega)^i. \tag{1.14}$$

The remainder term $\omega^{m+1}\Phi$ in equation (1.11) ensures the completeness on u [17], and it can be used to enforce additional constraints.

Using this generalized form, solution structures have been derived and catalog for most boundary conditions. Table 1.1 presents the solution structures for most popular boundary conditions for second order partial differential equation. Function ω in the solutions structure is constructed as an approximate distance field using R-functions. Depending on the desired computational properties, basis function can be selected form B-splines, polynomials, trigonometric polynomials, or any other popular choices.

Table 1.1: Example solution structures corresponding to boundary conditions for the second order partial differential equation

Type of Boundary Condition	Mathematical Formulation	Corresponding Solution Structure
Dirichlet	$u\|_{\partial\Omega} = \varphi$	$u = \omega\Phi + \varphi$
Neumann	$\frac{\partial u}{\partial n}\|_{\partial\Omega} = \varphi$	$u = \Phi - \omega D_1^{\omega}(\Phi) - \omega\Phi + \omega\varphi + \omega^2\Phi$
3-rd kind	$(\frac{\partial u}{\partial n} + hu)\|_{\partial\Omega} = \varphi$	$u = \Phi - \omega D_1^{\omega}(\Phi) - h\omega\Phi + \omega\varphi + \omega^2\Phi$
Mixed	$u\|_{\partial\Omega} = \varphi$ $(\frac{\partial u}{\partial n} + hu)\|_{\partial\Omega_2} = \psi$	$u = \omega_1\Phi + \frac{\omega_1\omega_2}{\omega_1+\omega_2}(\psi + \omega_2\Phi - D_1^{\omega_2}(\omega_1\Phi) - D_1^{\omega_2}(\varphi) - h\omega_1\Phi - h\varphi) + \varphi$

1.5 Personal Contribution

Many practical engineering analyses require solving nonlinear heat transfer problems caused by temperature-dependent material properties. These problems are usually solved only approximately by using one of the many known numerical techniques. In this thesis work, we extended the meshfree method with distance fields to support numerical solutions of such nonlinear transient heat transfer problems.

The thesis work involved developing and implementing numerical algorithms, data structure and software. Numerical algorithms were developed to solve nonlinear transient heat transfer problems with Dirichlet boundary conditions, and convective boundary conditions. Numerical formulation of the approximate solution of nonlinear heat transfer problems was based on Galerkin residual method. We applied Oseen and Newton-Kantorovich linearization schemes to linearize the nonlinear terms in nonlinear equations, which leads to an iterative procedure. We observed that Newton-Kantorovich scheme may not converge to the solution when temperature-dependence of thermal conductivity is very steep, and Oseen may be slow to converge to the solution. Therefore, we developed and implemented numerical algorithms that applied both schemes during the linearization process. This formulation produced an efficient linearization procedure which was observed to always converged to the solution.

The meshfree method with distance fields explicitly incorporates the temperature-dependent material properties in the approximate solution and it enables all prescribed boundary conditions to be satisfied exactly. Therefore, solving nonlinear transient heat transfer problems using the meshfree method with distance fields module that we developed, has the potential to produce fast converging and accurate results. The software was developed using C++ programming language. The computational methodology was validated by FEM.

1.6 Thesis Outline

The thesis is arranged as follows: Chapter 2 presents numerical formulations of solution to nonlinear transient heat transfer problems with Dirichlet boundary conditions and convective boundary conditions. The basis for the formulation to approximate solution of the nonlinear heat transfer problem is the Galerkin method [8], as a special case of the method of weighted residual. To solve the transient problem, a backward (implicit Euler) finite difference scheme is applied to the partial derivative of temperature with respect to time. The nonlinear equation is linearized by applying Oseen and Kantorovich linearization schemes [25].

In Chapter 3, results of the numerical experiments are discussed. Numerical experiments were conducted for both constant and temperature-dependent thermal conductivities. The behavior of the numerical algorithms are observed for temperature-dependence of thermal conductivity of real materials with actual temperature-dependency taken from standard materials database. Convergence and accuracy of the numerical solutions obtained by the meshfree method with distance fields are compared with analytical solutions, and solutions produced by commercial FEM package ANSYS 12.1. Chapter 4 discusses the benefits of using meshfree method with distance fields to solve nonlinear transient heat transfer problems, conclusion, and recommendations for future development of this project.

CHAPTER 2
NUMERICAL FORMULATION

2.1 Temperature-Dependent Thermal Conductivity

Solution of any field problem depends on the physical law governing the distribution of the physical quantities throughout the domain, boundary conditions describing the interaction of the domain and its external environment, and the initial conditions which determine the field at some point in time. The balance equation that described the transient heat conduction in solids with temperature-dependent thermal properties, thermal conductivity $\lambda(T)$, specific heat $c(T)$ and density $\rho(T)$, is given as a partial differential expression

$$c(T)\rho(T)\frac{\partial T}{\partial t} - div[\lambda(T)\nabla T] - Q = 0. \tag{2.1}$$

For this thesis work, specific heat and density are assumed to be constant, and there is no internal heat generation Q. The balanced equation (2.1) is therefore expressed as a nonlinear differential equation of the form

$$c\rho\frac{\partial T}{\partial t} - div[\lambda(T)\nabla T] = 0. \tag{2.2}$$

Equation (2.2) is a transient equation with three spacial coordinates (x, y, z). The approximate solution along these spacial coordinates are themselves function of time and their values at any time instant are dependent on the earlier solutions. The function describing the temperature distribution in space and time is presented as $T(x, y, z, t)$. To solve the problem, we first discretize by time, and then apply linearization scheme to obtain solution of the quasi-steady nonlinear problem.

2.2 Time stepping

We expressed the relationship between the temperatures and the rate of temperatures ($\frac{\partial T}{\partial t}$) at two different time instances , t_{n+1} and t_n, as

$$\frac{\partial T}{\partial t} = \frac{T^{n+1} - T^n}{\Delta t}, \tag{2.3}$$

where T^{n+1} expressed the temperature distributions at the current time t_{n+1} and T^n expressed the temperature distributions at the previous time t_n. Substituting expression (2.3) into equation (2.2) leads to two choices of finite difference time stepping schemes:

1. *Explicit (forward Euler) method.* The explicit method is very fast but requires small time steps to insure numerical stability. Applying this method to equation (2.2), we obtain

$$T^{n+1} = \frac{\Delta t}{c\rho}\lambda(T)\nabla^2 T^n + T^n. \tag{2.4}$$

2. *Implicit (backward Euler) method.* The implicit method requires much more computer storage than the explicit method but it has the advantage of using large time step, thus resulting in a more efficient procedure. Equation (2.2) for this method becomes

$$\frac{\Delta t}{c\rho}\lambda(T)\nabla^2 T^{n+1} - T^{n+1} = -T^n. \tag{2.5}$$

The backward implicit method is adopted and applied to solve the transient problem because this method is unconditionally stable, and it tends to eliminate oscillations in the solution. To complete the formulation of the nonlinear transient heat transfer problem we need to prescribe the initial conditions. We can apply known distribution of the temperature field, or assume that the initial temperature distribution occurred sufficiently far in advance, therefore, satisfies the steady-state version of the equation.

2.3 Galerkin Method for Modeling Heat Transfer Problems

A variety of numerical methods can be used to approximate the solution of equation (2.5). For this thesis, we used the weighted residual method known as Galerkin method [8] to obtain approximate solutions of nonlinear transient heat transfer problems. First, the balance equation (2.5) is written in the residual form as

$$r_B = \frac{\Delta t}{c\rho}\lambda(T)\nabla^2 T^{n+1} - T^{n+1} + T^n. \tag{2.6}$$

The Galerkin approach is formulated by describing the balance residual equation (2.6) as the weighted residual equation of the form

$$\int_\Omega r_B \omega \chi_j \, d\Omega = 0 \qquad j = 1, ..., N. \tag{2.7}$$

In equation (2.7), the residual r_B is multiplied by a test function $\omega\chi_j$, where ω is a distance field to the boundary $\partial\Omega$ and χ_i is a basis function. Substituting the balance residual expression (2.6) into equation (2.7), we obtain

$$\int_\Omega \frac{\Delta t}{c\rho}\lambda(T)\nabla^2 T^{n+1}(\omega\chi_j)\, d\Omega - \int_\Omega T^{n+1}(\omega\chi_j)\, d\Omega = -\int_\Omega T^n(\omega\chi_j)\, d\Omega. \tag{2.8}$$

Meshfree method with distance fields requires that solution of the boundary value problem equation (2.8) must incorporate the analytic information about the boundary conditions, as well as geometric information about the boundaries where these conditions are specified. As described in section 1.4.3, Rvachev proposed to represent the solution T^{n+1} we sought by a solution structure. A solution structure is a function that satisfies exactly all the prescribe boundary conditions. Solution structures corresponding to boundary conditions for heat transfer problems are well documented in Table 1.1.

According to [16], the solution structure (sought solution) can be split into homogeneous and nonhomogeneous parts:

$$T^{n+1} = T_0^{n+1} + T_1^{n+1}. \tag{2.9}$$

The homogeneous part T_0^{n+1} is a linear combination of basis function satisfying the homogeneous boundary conditions. The nonhomogeneous part T_1^{n+1} satisfies the boundary conditions, and contains the functions prescribed in the boundary conditions (1.2), (1.5) and (1.6). Substituting the two part solution structure (2.9) into equation (2.8) yields

$$\begin{aligned}\int_\Omega \frac{\Delta t}{c\rho}\nabla^2(T_0^{n+1} + T_1^{n+1})\lambda(T)(\omega\chi_j)\, d\Omega - \int_\Omega (T_0^{n+1} + T_1^{n+1})(\omega\chi_j)\, d\Omega = \\ -\int_\Omega T^n(\omega\chi_j)\, d\Omega.\end{aligned} \tag{2.10}$$

Expanding equation (2.10) and keeping only the homogeneous terms on the left side of the equations , we obtain

$$\begin{aligned}&\int_\Omega \frac{\Delta t}{c\rho}\nabla^2(T_0^{n+1})\lambda(T)(\omega\chi_j)\, d\Omega - \int_\Omega (T_0^{n+1})(\omega\chi_j)\, d\Omega = \\ &\quad -\int_\Omega \frac{\Delta t}{c\rho}\nabla^2(T_1^{n+1})\lambda(T)(\omega\chi_j)\, d\Omega + \int_\Omega (T_1^{n+1})(\omega\chi_j)\, d\Omega - \int_\Omega T^n(\omega\chi_j)\, d\Omega.\end{aligned} \tag{2.11}$$

At time t_{n+1}, equation (2.11) describes a quasi-steady nonlinear problem for the unknown temperature distribution T^{n+1}. Note that the right side of equation (2.11) contains only known quantities including the nonhomogeneous solution T_1^{n+1} and the temperature distribution T^n obtained at the previous time step t_n. The next step of solving equation (2.11) is to substitute for T_0^{n+1} and T_1^{n+1} the appropriate solution structure that defines the problem. The subsequent solution procedure will determine the coefficients that approximate the differential equation (2.2) and thus the temperature field T^{n+1}.

2.4 Dirichlet Boundary Conditions

The Dirichlet boundary conditions for nonlinear transient heat transfer is written as follows:

$$T_{|\partial\Omega} = T_1(x, y, z, t), \tag{2.12}$$

where $T_1(x, y, z, t)$ is the value of the temperature prescribed on the boundary $\partial\Omega$ of the geometric domain. The solution structure for nonhomogeneous Dirichlet boundary conditions is given as follows:

$$T^{n+1} = \omega\Phi^{n+1} + T_1^{n+1} \ , \tag{2.13}$$

where the product of the approximate distance to the boundary ω of the geometric domain and the unknown function Φ^{n+1} is the homogeneous part T_0^{n+1} of the two part solution structure (2.9). The function Φ^{n+1} cannot be determined exactly, therefore it is represented by linear combination of basis functions $\{\chi_i\}_{i=1}^{N}$

$$\Phi^{n+1} = \sum_{i=1}^{N} C_i^{n+1}\chi_i. \tag{2.14}$$

Substituting expression (2.14) into (2.13), the homogeneous part of the solution can be written as

$$T_0^{n+1} = \omega \sum_{i=1}^{N} C_i^{n+1}\chi_i. \tag{2.15}$$

The nonhomogeneous part T_1^{n+1} of the solution structure is given as

$$T_1^{n+1} = \varphi^{n+1} \tag{2.16}$$

and is the prescribed temperatures on the boundary $\partial\Omega$.

Substituting expressions (2.44) and (2.16) into equation (2.11) and computing for the unknown coefficients will yield approximate solution to the problem. But note that the first integral on both sides of equation (2.11) contains second derivatives of temperatures. Balancing the order of the differentiation by shifting one derivative from temperature to the test function $(\omega\chi_j)$ in these integrals has proven to be beneficial. We can do this by applying Gauss (divergence) theorem to these integral which yields:

$$\begin{aligned}\int_\Omega \frac{\Delta t}{c\rho}\nabla^2(T_0^{n+1})\lambda(T)(\omega\chi_j)\,d\Omega =& \\ -\int_\Omega \frac{\Delta t}{c\rho}\nabla(T_0^{n+1})\lambda(T)\nabla(\omega\chi_j)\,d\Omega &+ \int_{\partial\Omega}\frac{\Delta t}{c\rho}\frac{\partial(T_0^{n+1})}{\partial n}\lambda(T)(\omega\chi_j)\,dS \\ & j = 1, ..., N\end{aligned} \tag{2.17}$$

and

$$\begin{aligned}-\int_\Omega \frac{\Delta t}{c\rho}\nabla^2(T_1^{n+1})\lambda(T)(\omega\chi_j)\,d\Omega =& \\ \int_\Omega \frac{\Delta t}{c\rho}\nabla(T_1^{n+1})\lambda(T)\nabla(\omega\chi_j)\,d\Omega &- \int_{\partial\Omega}\frac{\Delta t}{c\rho}\frac{\partial(T_1^{n+1})}{\partial n}\lambda(T)(\omega\chi_j)\,dS \\ & j = 1, ..., N.\end{aligned} \tag{2.18}$$

Since ω vanishes on the boundary $\partial\Omega$, the boundary integrals in (2.17) and (2.18) become zero. Substituting the results of (2.17) and (2.18) into equation (2.11) yields:

$$\begin{aligned}-\int_\Omega \frac{\Delta t}{c\rho}\nabla(T_0^{n+1})\lambda(T)\nabla(\omega\chi_j)\,d\Omega - \int_\Omega (T_0^{n+1})(\omega\chi_j)\,d\Omega =& \\ \int_\Omega \frac{\Delta t}{c\rho}\nabla(T_1^{n+1})\lambda(T)\nabla(\omega\chi_j)\,d\Omega + \int_\Omega (T_1^{n+1})(\omega\chi_j)\,d\Omega &- \int_\Omega T^n(\omega\chi_j)\,d\Omega \\ & j = 1, ..., N.\end{aligned} \tag{2.19}$$

Equation (2.19) is now a single weighted residual equation which contains the solution to nonlinear transient heat transfer problems with Dirichlet boundary conditions. Since the meshfree method with distance fields solution structure satisfies the given boundary conditions exactly, solving the problem entails computing the set of unknown coefficients $\{C_i^{n+1}\}_{i=1}^{N}$ in the solution structure that gives the best approximation to the differential equation (2.2). However, equation (2.19) contains the nonlinear terms that have to be linearized before solution to the problem can be found.

Since the way of linearization can significantly affect the rate of convergence towards the final solution, choice of an appropriate linearization method is important. In this thesis work, we employed two different linearization schemes to linearize the nonlinear terms in equation (2.19); Oseen linearization scheme and Newton-Kantorovich linearization scheme. Both Oseen and Newton-Kantorovich linearization schemes lead to an iterative solution procedure. Since the problem is also transient, the numerical algorithm requires two loops:

1. Loop that solves the quasi-steady nonlinear problem at the current time t_{n+1} by updating the nonlinear terms until solution converges. In the derivation that follows, superscripts $k+1$ and k denote solutions at the current and previous iterations respectively for this loop.
2. Loop that propagates the converged solution in time Δt.

2.4.1 Oseen Linearization

Oseen linearization is the simplest linearization scheme used to linearize the nonlinear terms in equation (2.19). It leads to an iterative procedure that involves updating the nonlinear terms with values evaluated using solutions obtained at the previous iteration counter. Updating the terms continues until the difference between two consecutive solutions is sufficiently small. Applying Oseen linearization scheme to equation (2.19), we obtain

$$\begin{aligned}\int_\Omega \frac{\Delta t}{c\rho}\nabla(T_0^{k+1})\lambda(T^k)\nabla(\omega\chi_j)\,d\Omega - \int_\Omega \nabla(T_0^{k+1})(\omega\chi_j)\,d\Omega = \\ \int_\Omega \frac{\Delta t}{c\rho}\nabla(T_1^{n+1})\lambda(T^k)\nabla(\omega\chi_j)\,d\Omega + \int_\Omega (T_1^{n+1})(\omega\chi_j)\,d\Omega - \int_\Omega T^k(\omega\chi_j)\,d\Omega\end{aligned} \tag{2.20}$$

Note that T^k in equation (2.20) is the solution obtained at kth iteration during the iterative procedure that linearizes the nonlinear terms. The nonhomogeneous solution T_1^{n+1} is evaluated at the current time step but it is not temperature-dependent.

Equation (2.20) is a system of linear algebraic equations $[a_{ij}][C_i^{k+1}] = [b_j]$ whose solution gives the numerical values of the unknown coefficients C_i^{k+1} in the solution structure. The unknown coefficients C_i^{k+1} are computed as follows:

$$\begin{aligned} a_{i,j} &= -\frac{\Delta t}{c\rho}\int_\Omega \nabla(\omega\chi_i)\lambda(T^k)\nabla(\omega\chi_j)\,d\Omega - \int_\Omega (\omega\chi_i)(\omega\chi_j)\,d\Omega \\ b_j &= \frac{\Delta t}{c\rho}\int_\Omega \nabla(T_1^{n+1})\lambda(T^k)\nabla(\omega\chi_j)\,d\Omega + \int_\Omega (T_1^{n+1})(\omega\chi_j)\,d\Omega - \int_\Omega T^k(\omega\chi_j)\,d\Omega \end{aligned} \tag{2.21}$$

Solving the linear system (2.55) and substituting the value of coefficients C_i^{k+1} into the solution structure (2.13) yields an approximate solution T^{n+1} to the quasi-steady nonlinear problem at time t_{n+1}.

2.4.2 Newton-Kantorovich Linearization

Applying Newton-Kantorovich linearization scheme to equation (2.19) leads to and iterative procedure. The procedure involves updating the nonlinear terms with values evaluated using solutions obtained at the previous iteration counter. Updating the terms continues until the difference between the two consecutive solutions is sufficiently small. Newton-Kantorovich linearization enjoys rapid convergence and has been successfully applied to solve for nonlinear equations in fluid dynamics and heat transfer [25].

Consider a function uv, we can expand it in a Taylor series about the current value and terminate the series expansion after the first-derivative terms. As shown in

[25], the result is as follows:

$$\begin{aligned}
u^{k+1}v^{k+1} &= u^k v^k + \left[\frac{\partial}{\partial u}(uv)^k\right](u^{k+1}-u^k) + \left[\frac{\partial}{\partial v}(uv)^k\right](v^{k+1}-v^k) + H.O.T \\
&= u^k v^k + v^k(u^{k+1}-u^k) + u^k(v^{k+1}-v^k) \\
&= u^k v^k + u^{k+1}v^k - u^k v^k + u^k v^{k+1} - u^k v^k \\
u^{k+1}v^{k+1} &= u^{k+1}v^k + u^k v^{k+1} - u^k v^k
\end{aligned} \tag{2.22}$$

Linearization of the temperature-dependent terms in equation (2.19) by Newton-Kantorovich scheme, starts from rewriting equation (2.19) as

$$\begin{aligned}
&-\int_\Omega \frac{\Delta t}{c\rho}\nabla(T_0^{k+1})\lambda(T^{k+1})\nabla(\omega\chi_j)\,d\Omega - \int_\Omega (T_0^{k+1})(\omega\chi_j)\,d\Omega = \\
&\quad \int_\Omega \frac{\Delta t}{c\rho}\nabla(T_1^{k+1})\lambda(T^{k+1})\nabla(\omega\chi_j)\,d\Omega + \int_\Omega (T_1^{n+1})(\omega\chi_j)\,d\Omega - \int_\Omega T^k(\omega\chi_j)\,d\Omega \\
&\qquad j = 1, ..., N.
\end{aligned} \tag{2.23}$$

According to (2.22), we can rewrite the terms $\nabla(T_0^{k+1})\lambda(T^{k+1})$ and $\nabla(T_1^{k+1})\lambda(T^{k+1})$ in equation (2.23) as:

$$\nabla T_0^{k+1}\lambda(T^{k+1}) = \nabla T_0^{k+1}\lambda(T^k) + \nabla T_0^k\lambda(T^{k+1}) - \nabla T_0^k\lambda(T^k) \tag{2.24}$$

and

$$\nabla T_1^{k+1}\lambda(T^{k+1}) = \nabla T_1^{k+1}\lambda(T^k) + \nabla T_1^k\lambda(T^{k+1}) - \nabla T_1^k\lambda(T^k) \tag{2.25}$$

Since Newton-Kantorovich linearization strategy is to evaluate the nonlinear terms using solutions obtained at the previous iteration counter k, we expressed $\lambda(T^{k+1})$ as a Taylor series expansion

$$\lambda(T^{k+1}) = \lambda(T^k) + \frac{\partial\lambda}{\partial T^k}(T^{k+1} - T^k). \tag{2.26}$$

Expanding expression (2.26) and substituting into expressions (2.24) and (2.25) yields:

$$\nabla T_0^{k+1}\lambda(T^{k+1}) = \nabla T_0^{k+1}\lambda(T^k) + \nabla T_0^k \frac{\partial\lambda}{\partial T^k} T^{k+1} - \nabla T_0^k \frac{\partial\lambda}{\partial T^k} T^k \tag{2.27}$$

and

$$\nabla T_1^{k+1}\lambda(T^{k+1}) = \nabla T_1^{k+1}\lambda(T^k) + \nabla T_1^k \frac{\partial\lambda}{\partial T^k} T^{k+1} - \nabla T_1^k \frac{\partial\lambda}{\partial T^k} T^k \tag{2.28}$$

In expressions (2.27) and (2.28), the nonlinear terms are now expressed as functions of the temperature distribution obtained at the previous iteration T^k. Substituting expressions (2.27) and (2.28) into equation (2.23), we obtained the single weighted residual equation with Newton-Kantorovich linearization scheme applied as

$$\begin{aligned}
&-\int_\Omega \frac{\Delta t}{c\rho}\nabla(T_0^{k+1})\lambda(T^k)\nabla(\omega\chi_j)\,d\Omega - \int_\Omega \frac{\Delta t}{c\rho}\nabla(T_0^k)\frac{\partial\lambda}{\partial T^k}T^{k+1}\nabla(\omega\chi_j)\,d\Omega \\
&+\int_\Omega \frac{\Delta t}{c\rho}\nabla(T_0^k)\frac{\partial\lambda}{\partial T^k}T^k\nabla(\omega\chi_j)\,d\Omega - \int_\Omega (T_0^{k+1})(\omega\chi_j)\,d\Omega = \\
&\int_\Omega \frac{\Delta t}{c\rho}\nabla(T_1^{n+1})\lambda(T^k)\nabla(\omega\chi_j)\,d\Omega + \int_\Omega \frac{\Delta t}{c\rho}\nabla(T_1^k)\frac{\partial\lambda}{\partial T^k}T^{k+1}\nabla(\omega\chi_j)\,d\Omega \\
&-\int_\Omega \frac{\Delta t}{c\rho}\nabla(T_1^k)\frac{\partial\lambda}{\partial T^k}T_1^k\nabla(\omega\chi_j)\,d\Omega + \int_\Omega (T_1^{n+1})(\omega\chi_j)\,d\Omega - \int_\Omega T^k(\omega\chi_j)\,d\Omega \\
&\qquad\qquad\qquad\qquad j = 1, ..., N.
\end{aligned} \tag{2.29}$$

We can express T^{k+1} as a two part homogeneous and nonhomogeneous solution structure $T^{k+1} = T_0^{k+1} + T_1^{k+1}$ which yields:

$$\begin{aligned}
&-\int_\Omega \frac{\Delta t}{c\rho}\nabla(T_0^{k+1})\lambda(T^k)\nabla(\omega\chi_j)\,d\Omega - \int_\Omega \frac{\Delta t}{c\rho}(T_0^{k+1})\frac{\partial\lambda}{\partial T^k}\nabla(T^k)\nabla(\omega\chi_j)\,d\Omega \\
&-\int_\Omega (T_0^{k+1})(\omega\chi_j)\,d\Omega = \\
&\int_\Omega \frac{\Delta t}{c\rho}\nabla(T_1^{n+1})\lambda(T^k)\nabla(\omega\chi_j)\,d\Omega + \int_\Omega \frac{\Delta t}{c\rho}\nabla(T^k)\frac{\partial\lambda}{\partial T^k}(T_1^{n+1})\nabla(\omega\chi_j)\,d\Omega \\
&-\int_\Omega \frac{\Delta t}{c\rho}\nabla(T^k)\frac{\partial\lambda}{\partial T^k}(T^k)\nabla(\omega\chi_j)\,d\Omega + \int_\Omega (T_1^{n+1})(\omega\chi_j)\,d\Omega - \int_\Omega T^k(\omega\chi_j)\,d\Omega \\
&\qquad\qquad\qquad\qquad j = 1, ..., N.
\end{aligned} \tag{2.30}$$

Note that the nonhomogeneous solution T_1^{n+1} is evaluated at the current time step but it is not temperature-dependent, therefore it is denoted by the superscript $n+1$.

Equation (2.30) is a system of linear algebraic equations $[a_{ij}][C_i^{k+1}] = [b_j]$ whose solution gives the numerical values of the unknown coefficients in the solution structure. The unknown coefficients C_i^{k+1} are computed as follows:

$$\begin{aligned} a_{i,j} &= -\int_\Omega \frac{\Delta t}{c\rho}\nabla(\omega\chi_i)\lambda(T^k)\nabla(\omega\chi_j)\,d\Omega - \int_\Omega \frac{\Delta t}{c\rho}(\omega\chi_i)\frac{\partial\lambda}{\partial T^k}\nabla T^k\nabla(\omega\chi_j)\,d\Omega \\ &\quad - \int_\Omega (\omega\chi_i)(\omega\chi_j)\,d\Omega \\ b_j &= \int_\Omega \frac{\Delta t}{c\rho}\nabla(T_1^{n+1})\lambda(T^k)\nabla(\omega\chi_j)\,d\Omega + \int_\Omega \frac{\Delta t}{c\rho}(T_1^{n+1})\frac{\partial\lambda}{\partial T^k}\nabla T^k\nabla(\omega\chi_j)\,d\Omega \\ &\quad - \int_\Omega \frac{\Delta t}{c\rho}\nabla(T^k)\frac{\partial\lambda}{\partial T^k}T^k\nabla(\omega\chi_j)\,d\Omega + \int_\Omega T_1^{n+1}(\omega\chi_j)\,d\Omega - \int_\Omega T^k(\omega\chi_j)\,d\Omega \end{aligned} \tag{2.31}$$

Solving the linear system (2.65) and substituting the value of coefficients C_i^{k+1} into the solution structure (2.13) yields an approximate solution T^{n+1} to the quasi-steady nonlinear problem at time t_{n+1}.

2.5 Convective Boundary Conditions

Nonlinear convective boundary conditions has the general form

$$\lambda(T)\frac{\partial T}{\partial n}\bigg|_{\partial\Omega} = -\alpha(T - T_{env}), \tag{2.32}$$

where $\lambda(T)$ is temperature-dependent thermal conductivity, T_{env} is temperature of the medium surrounding the convective boundary, and α is heat transfer coefficient. We can rewrite (2.32) as

$$\left(\frac{\partial T}{\partial n} + \frac{\alpha}{\lambda(T)}T\right)\bigg|_{\partial\Omega} = \frac{\alpha T_{env}}{\lambda(T)} \tag{2.33}$$

and expressed the following terms in (2.33) as

$$\frac{\alpha}{\lambda(T)} = h \tag{2.34}$$

$$\frac{\alpha T_{env}}{\lambda(T)} = \varphi \tag{2.35}$$

Substituting expressions (2.34) and (2.35) into (2.32), we can rewrite the nonlinear convective boundary condition as

$$\left.\frac{\partial T}{\partial n}\right|_{\partial\Omega} = \varphi - hT\Big|_{\partial\Omega}. \tag{2.36}$$

According to [16], the solution structure for a boundary value problem with any types boundary conditions can be represented by the generalized Taylor series expansion of the form

$$T = T(0) + \omega T'(0) + \omega^2 T''(0) + (\omega^3)O, \tag{2.37}$$

where $T(0) = T|_{\partial\Omega}$ and $T(0)' = \frac{\partial T}{\partial n}|_{\partial\Omega}$.

From this generalized representation of the solution structure, the corresponding solution structure for boundary value problems with convective boundary conditions (2.36) can be written in the form

$$T = \Phi_1 - \omega D_1^{\omega}(\Phi) + \omega(\varphi - h\Phi_1) + \omega^2\Phi_2, \tag{2.38}$$

where

$$D_1^{\omega}(\Phi) = \frac{\partial\omega}{\partial x}\frac{\partial\Phi}{\partial x} + \frac{\partial\omega}{\partial y}\frac{\partial\Phi}{\partial y} + \frac{\partial\omega}{\partial z}\frac{\partial\Phi}{\partial z} = \nabla\omega\cdot\nabla\Phi \tag{2.39}$$

is a differential operator in the direction of the internal normal to the boundary $\partial\Omega$. It should be noted clearly that the remainder term $\omega^2\Phi_2$ assures completeness of this solution structure [17]. The solution structure representing the sought solution T^{n+1} of our nonlinear transient heat transfer problem with convective boundary conditions can be written as

$$T^{n+1} = \Phi^{n+1} - \omega D_1^{\omega}(\Phi^{n+1}) - \omega h\Phi^{n+1} + \omega\varphi^{n+1}. \tag{2.40}$$

As shown in [16], it is convenient to represent (2.40) as sum $T^{n+1} = T_0^{n+1} + T_1^{n+1}$ of homogeneous and nonhomogeneous parts:

$$T_0^{n+1} = \Phi^{n+1} - \omega D_1^{\omega}(\Phi^{n+1}) - \omega h \Phi^{n+1} \tag{2.41}$$

$$T_1^{n+1} = \omega \varphi^{n+1}. \tag{2.42}$$

The function Φ^{n+1} cannot be determined exactly, therefore it is represented by linear combination of basis functions $\{\chi_i\}_{i=1}^{N}$

$$\Phi^{n+1} = \sum_{i=1}^{N} C_i^{n+1} \chi_i \tag{2.43}$$

Substituting expression (2.43) into (2.40), the homogeneous part of the solution can be written as

$$T_0^{n+1} = \sum_{i=1}^{N} C_i^{n+1} \left(\chi_i - D_1^{\omega} \chi_i + h \chi_i \right) \omega \tag{2.44}$$

and further represented as

$$T_0^{n+1} = \sum_{i=1}^{N} C_i^{n+1} (\xi_i) \tag{2.45}$$

Substituting expressions (2.42) and (2.45) into equation (2.11) and computing for the unknown coefficients will yield approximate solution to the problem. But note that the first integral on both sides of equation (2.11) contains second derivatives of temperatures. Balancing the order of the differentiation by shifting one derivative from temperature to the test function ($\omega \chi_j$) in these integrals has proven to be beneficial. We can do this by applying Gauss (divergence) theorem to these integral.

Applying Gauss (divergence) theorem to the first integral on the left side of equation(2.11)) yields:

$$\begin{aligned}\int_{\Omega} \frac{\Delta t}{c\rho} \nabla^2(T_0^{n+1})\lambda(T)\xi_j \, d\Omega = \\ -\int_{\Omega} \frac{\Delta t}{c\rho} \nabla(T_0^{n+1})\lambda(T)\nabla\xi_j \, d\Omega + \int_{\partial\Omega} \frac{\Delta t}{c\rho} \frac{\partial(T_0^{n+1})}{\partial n}\lambda(T)\xi_j \, dS\end{aligned} \tag{2.46}$$

From equation (2.33) we can write the homogeneous convective boundary conditions as

$$\left(\frac{\partial T_0^{n+1}}{\partial n} + \frac{\alpha}{\lambda(T)} T_0^{n+1}\right)\Bigg|_{\partial\Omega} = 0 \tag{2.47}$$

$$\frac{\partial T_0^{n+1}}{\partial n}\Bigg|_{\partial\Omega} = -\frac{\alpha}{\lambda(T)} T_0^{n+1}\Bigg|_{\partial\Omega} \tag{2.48}$$

Substituting expression (2.48 into (2.46) gives

$$\begin{aligned}\int_{\Omega} \frac{\Delta t}{c\rho} \nabla^2(T_0^{n+1})\lambda(T)\xi_j \, d\Omega = \\ -\int_{\Omega} \frac{\Delta t}{c\rho} \nabla(T_0^{n+1})\lambda(T)\nabla\xi_j \, d\Omega - \int_{\partial\Omega} \frac{\Delta t}{c\rho} T_0^{n+1}\alpha\xi_j \, dS\end{aligned} \tag{2.49}$$

Applying Gauss (divergence) theorem to the first integral on the right side of equation(2.11)) yields:

$$\begin{aligned}-\int_{\Omega} \frac{\Delta t}{c\rho} \nabla^2(T_1^{n+1})\lambda(T)\xi_j \, d\Omega = \\ \int_{\Omega} \frac{\Delta t}{c\rho} \nabla(T_1^{n+1})\lambda(T)\nabla\xi_j \, d\Omega - \int_{\partial\Omega} \frac{\Delta t}{c\rho} \frac{\partial(T_1^{n+1})}{\partial n}\lambda(T)\xi_j \, dS\end{aligned} \tag{2.50}$$

From equation (2.33) we can write the nonhomogeneous convective boundary condition as

$$\frac{\partial T_1^{n+1}}{\partial n}\Bigg|_{\partial\Omega} = \frac{\alpha T_{env}}{\lambda(T)}\Bigg|_{\partial\Omega} \tag{2.51}$$

Substituting expression (2.51 into (2.46) we obtain,

$$
\begin{aligned}
-\int_\Omega \frac{\Delta t}{c\rho}\nabla^2(T_1^{n+1})\lambda(T)\xi_j\,d\Omega = \\
\int_\Omega \frac{\Delta t}{c\rho}\nabla(T_1^{n+1})\lambda(T)\nabla\xi_j\,d\Omega - \int_{\partial\Omega}\frac{\Delta t}{c\rho}\alpha T_{env}\xi_j\,dS
\end{aligned}
\tag{2.52}
$$

We now substitute (2.46) and (2.46) into (2.11) which yields:

$$
\begin{aligned}
&-\int_\Omega \frac{\Delta t}{c\rho}\nabla T_0^{n+1}\lambda(T)\nabla\xi_j\,d\Omega - \int_{\partial\Omega}\frac{\Delta t}{c\rho}T_0^{n+1}\alpha\xi_j\,dS \\
&-\int_\Omega T_0^{n+1}\xi_j\,d\Omega = \\
&\int_\Omega \frac{\Delta t}{c\rho}\nabla T_1^{n+1}\lambda(T)\nabla\xi_j\,d\Omega - \int_{\partial\Omega}\frac{\Delta t}{c\rho}\alpha T_{env}\xi_j\,dS \\
&+\int_\Omega T_1^{n+1}\xi_j\,d\Omega - \int_\Omega T^n\xi_j\,d\Omega \\
&\qquad\qquad\qquad\qquad\qquad\qquad j = 1, ..., N.
\end{aligned}
\tag{2.53}
$$

Equation (2.53) is now a single weighted residual equation which contains the solution to nonlinear transient heat transfer problems with convective boundary conditions. Solving the problem entails computing the set of unknown coefficients $\{C_i^{n+1}\}_{i=1}^N$ in the solution structure that gives the best approximation to the differential equation (2.2). However, equation (2.53) contains the nonlinear terms that have to be linearized before solution to the problem can be found.

Since the way of linearization can significantly affect the rate of convergence towards the final solution, choice of an appropriate linearization method is important. In this thesis work, we employed two different linearization schemes to linearize the nonlinear terms in equation (2.53); Oseen linearization scheme and Newton-Kantorovich linearization scheme. Both Oseen and Newton-Kantorovich linearization schemes lead to an iterative solution procedure. Since the problem is also transient, the numerical algorithm requires two loops:

1. Loop that solves the quasi-steady nonlinear problem at the current time t_{n+1} by

updating the nonlinear terms until solution converges. In the derivation that follows, superscripts $k + 1$ and k denote solutions at the current and previous iterations respectively for this loop.

2. Loop that propagates the converged solution in time Δt.

2.5.1 Oseen Linearization

Application of Oseen linearization to linearize the nonlinear terms in equation (2.53) leads to an iterative solution procedure. The procedure involves updating the nonlinear terms with values evaluated using solutions obtained at the previous iteration counter. Updating the nonlinear terms continues until the difference between the current and previous solutions is sufficiently small. Applying Oseen linearization scheme to equation (2.53) yields

$$
\begin{aligned}
& -\int_{\Omega} \frac{\Delta t}{c\rho} \nabla T_0^{n+1} \lambda(T) \nabla \xi_j \, d\Omega - \int_{\partial\Omega} \frac{\Delta t}{c\rho} T_0^{n+1} \alpha \xi_j \, dS \\
& -\int_{\Omega} T_0^{n+1} \xi_j \, d\Omega = \\
& \int_{\Omega} \frac{\Delta t}{c\rho} \nabla T_1^{n+1} \lambda(T) \nabla \xi_j \, d\Omega - \int_{\partial\Omega} \frac{\Delta t}{c\rho} \alpha T_{env} \xi_j \, dS \\
& +\int_{\Omega} T_1^{n+1} \xi_j \, d\Omega - \int_{\Omega} T^k \xi_j \, d\Omega \\
& \qquad\qquad\qquad\qquad\qquad\qquad j = 1, ..., N.
\end{aligned}
\tag{2.54}
$$

Note that T^k in equation (2.54) is the solution obtained at kth iteration during the iterative procedure that linearize the nonlinear terms. The nonhomogeneous solution T_1^{n+1} is evaluated at the current time step but it is not temperature-dependent.

Equation (2.54) is a system of linear algebraic equations $[a_{ij}][C_i^{k+1}] = [b_j]$ whose solution gives the numerical values of the unknown coefficients in the solution structure.

The unknown coefficients C_i^{k+1} are computed as follows:

$$\begin{aligned} a_{i,j} &= -\frac{\Delta t}{c\rho}\int_\Omega \nabla\xi_i\lambda(T^k)\nabla\xi_j\, d\Omega - \int_\Omega \xi_i\xi_j\, d\Omega \\ &- \int_{\partial\Omega}\frac{\Delta t}{c\rho}\xi_i\alpha\xi_j\, dS \\ b_j &= \frac{\Delta t}{c\rho}\int_\Omega \nabla T_1^{n+1}\lambda(T^k)\nabla\xi_j\, d\Omega + \int_\Omega T_1^{n+1}\xi_j\, d\Omega - \int_\Omega T^k\xi_j\, d\Omega \\ &- \int_{\partial\Omega}\frac{\Delta t}{c\rho}T_{env}\alpha\xi_j\, dS \end{aligned} \tag{2.55}$$

Solving the linear system (2.55) and substituting the value of coefficients C_i^{k+1} into the solution structure (2.40) yields an approximate solution T^{n+1} to the quasi-steady nonlinear problem at time t_{n+1}.

2.5.2 Newton-Kantorovich Linearization

Applying Newton-Kantorovich linearization scheme to equation (2.53) leads to and iterative procedure. The procedure involves updating the nonlinear terms with values evaluated using solutions obtained at the previous iteration counter. Updating the term continues until the difference between the current and previous solutions is sufficiently small. Newton-Kantorovich linearization enjoys rapid convergence and has been successfully applied to solve for nonlinear equations in fluid dynamics and heat transfer [25].

Consider a function uv, we can expand it in a Taylor series about the current value and terminate the series expansion after the first-derivative terms. As shown in [25], the result is as follows:

$$\begin{aligned} u^{k+1}v^{k+1} &= u^kv^k + \left[\frac{\partial}{\partial u}(uv)^k\right](u^{k+1}-u^k) + \left[\frac{\partial}{\partial v}(uv)^k\right](v^{k+1}-v^k) + H.O.T \\ &= u^{k+1}v^k + u^kv^{k+1} - u^kv^k \end{aligned} \tag{2.56}$$

Linearization of the nonlinear terms in equation (2.53) by Newton-Kantorovich scheme, starts from rewriting equation (2.53) as

$$\begin{aligned}
&-\int_\Omega \frac{\Delta t}{c\rho}\nabla T_0^{k+1}\lambda(T^{k+1})\nabla\xi_j\, d\Omega - \int_{\partial\Omega}\frac{\Delta t}{c\rho}T_0^{n+1}\alpha\xi_j\, dS \\
&-\int_\Omega T_0^{n+1}\xi_j\, d\Omega = \\
&\int_\Omega \frac{\Delta t}{c\rho}\nabla T_1^{k+1}\lambda(T^{k+1})\nabla\xi_j\, d\Omega - \int_{\partial\Omega}\frac{\Delta t}{c\rho}\alpha T_{env}\xi_j\, dS \qquad (2.57)\\
&+\int_\Omega T_1^{n+1}\xi_j\, d\Omega - \int_\Omega T^k\xi_j\, d\Omega \\
&\qquad\qquad\qquad\qquad j = 1, ..., N.
\end{aligned}$$

According to (2.56), we can rewrite $\nabla T_0^{n+1}\lambda(T^{k+1})$ and $\nabla(T_1^{k+1})\lambda(T^{k+1})$ in equation (2.57) as:

$$\nabla T_0^{k+1}\lambda(T^{k+1}) = \nabla T_0^{k+1}\lambda(T^k) + \nabla T_0^k\lambda(T^{k+1}) - \nabla T_0^k\lambda(T^k) \qquad (2.58)$$

and

$$\nabla T_1^{k+1}\lambda(T^{k+1}) = \nabla T_1^{k+1}\lambda(T^k) + \nabla T_1^k\lambda(T^{k+1}) - \nabla T_1^k\lambda(T^k) \qquad (2.59)$$

Since Newton-Kantorovich linearization strategy is to evaluate the nonlinear terms using solutions obtained at the previous iteration counter k, we expressed $\lambda(T^{k+1})$ as a Taylor series expansion

$$\lambda(T^{k+1}) = \lambda(T^k) + \frac{\partial\lambda}{\partial T^k}(T^{k+1} - T^k). \qquad (2.60)$$

Expanding expression (2.60) and substituting it into expressions (2.58) and (2.59) yields:

$$\nabla T_0^{k+1}\lambda(T^{k+1}) = \nabla T_0^{k+1}\lambda(T^k) + \nabla T_0^k\frac{\partial\lambda}{\partial T^k}T^{k+1} - \nabla T_0^k\frac{\partial\lambda}{\partial T^k}T^k \qquad (2.61)$$

and

$$\nabla T_1^{k+1}\lambda(T^{k+1}) = \nabla T_1^{k+1}\lambda(T^k) + \nabla T_1^k\frac{\partial\lambda}{\partial T^k}T^{k+1} - \nabla T_1^k\frac{\partial\lambda}{\partial T^k}T^k \qquad (2.62)$$

In expressions (2.61) and (2.62), the nonlinear terms are now expressed as functions of the temperature distribution obtained at the previous iteration T^k. Substituting expressions (2.61) and (2.62) into equation (2.57), we obtain the single weighted residual equation with Newton-Kantorovich linearization scheme applied as

$$\begin{aligned}
&-\int_\Omega \frac{\Delta t}{c\rho}\nabla T_0^{n+1}\lambda(T^k)\nabla\xi_j\,d\Omega - \int_\Omega \frac{\Delta t}{c\rho}\nabla T_0^k\frac{\partial\lambda}{\partial T^k}T^{k+1}\nabla\xi_j\,d\Omega \\
&+\int_\Omega \frac{\Delta t}{c\rho}\nabla T_0^k\frac{\partial\lambda}{\partial T^k}T^k\nabla\xi_j\,d\Omega - \int_\Omega T_0^{n+1}\xi_j\,d\Omega \\
&-\int_{\partial\Omega}\frac{\Delta t}{c\rho}T_0^{n+1}\alpha\xi_j\,dS = \\
&\int_\Omega \frac{\Delta t}{c\rho}\nabla T_1^{n+1}\lambda(T^k)\nabla\xi_j\,d\Omega + \int_\Omega \frac{\Delta t}{c\rho}\nabla T_1^k\frac{\partial\lambda}{\partial T^k}T^{k+1}\nabla\xi_j\,d\Omega \\
&-\int_\Omega \frac{\Delta t}{c\rho}\nabla T_1^k\frac{\partial\lambda}{\partial T^k}T_1^k\nabla\xi_j\,d\Omega + \int_\Omega T_1^{n+1}\xi_j\,d\Omega - \int_\Omega T^k\xi_j\,d\Omega \\
&-\int_{\partial\Omega}\frac{\Delta t}{c\rho}\alpha T_{env}\xi_j\,dS \\
&\qquad\qquad\qquad\qquad j = 1, ..., N.
\end{aligned} \tag{2.63}$$

We can express T^{k+1} as a two part homogeneous and nonhomogeneous solution structure $T^{k+1} = T_0^{k+1} + T_1^{k+1}$. Substituting the two part solution structure for T^{k+1} into equation (2.29) and keeping the homogeneous solution T_0^{k+1} terms on the left side of the equation yields:

$$\begin{aligned}
&-\int_\Omega \frac{\Delta t}{c\rho}\nabla T_0^{n+1}\lambda(T^k)\nabla\xi_j\,d\Omega - \int_\Omega \frac{\Delta t}{c\rho}T_0^{n+1}\frac{\partial\lambda}{\partial T^k}\nabla T^k\nabla\xi_j\,d\Omega \\
&-\int_\Omega T_0^{n+1}\xi_j\,d\Omega - \int_{\partial\Omega}\frac{\Delta t}{c\rho}T_0^{n+1}\alpha\xi_j\,dS = \\
&\int_\Omega \frac{\Delta t}{c\rho}T_1^{n+1}\lambda(T^k)\nabla\xi_j\,d\Omega + \int_\Omega \frac{\Delta t}{c\rho}\nabla T^k\frac{\partial\lambda}{\partial T^k}T_1^{n+1}\nabla\xi_j\,d\Omega \\
&-\int_\Omega \frac{\Delta t}{c\rho}\nabla T^k\frac{\partial\lambda}{\partial T^k}T^k\nabla\xi_j\,d\Omega + \int_\Omega T_1^{n+1}\xi_j\,d\Omega - \int_\Omega T^n\xi_j\,d\Omega \\
&-\int_{\partial\Omega}\frac{\Delta t}{c\rho}\alpha T_{env}\xi_j\,dS \\
&\qquad\qquad\qquad\qquad j = 1, ..., N.
\end{aligned} \tag{2.64}$$

Note that the nonhomogeneous solution T_1^{n+1} is evaluated at the current time step but it is not temperature-dependent.

Equation (2.64) is a system of linear algebraic equations $[a_{ij}][C_i^{k+1}] = [b_j]$ whose solution gives the numerical values of the unknown coefficients in the solution structure. The unknown coefficients C_i^{k+1} are computed as follows:

$$\begin{aligned}
a_{i,j} = &-\int_\Omega \frac{\Delta t}{c\rho}\nabla\xi_i\lambda(T^k)\nabla\xi_j\, d\Omega - \int_\Omega \frac{\Delta t}{c\rho}\xi_i\frac{\partial\lambda}{\partial T^k}\nabla T^k\nabla\xi_j\, d\Omega \\
&-\int_\Omega \xi_i\xi_j\, d\Omega - \int_{\partial\Omega}\frac{\Delta t}{c\rho}\xi_i\alpha\xi_j\, dS \\
b_j = &\int_\Omega \frac{\Delta t}{c\rho}\nabla T_1^{n+1}\lambda(T^k)\nabla\xi_j\, d\Omega + \int_\Omega \frac{\Delta t}{c\rho}T_1^{n+1}\frac{\partial\lambda}{\partial T^k}\nabla T^k\nabla\xi_j\, d\Omega \\
&-\int_\Omega \frac{\Delta t}{c\rho}\nabla T^k\frac{\partial\lambda}{\partial T^k}T^k\nabla\xi_j\, d\Omega + \int_\Omega T_1^{n+1}\xi_j\, d\Omega - \int_\Omega T^n\xi_j\, d\Omega \\
&-\int_{\partial\Omega}\frac{\Delta t}{c\rho}T_{env}\alpha\xi_j\, dS
\end{aligned} \tag{2.65}$$

Solving the linear system (2.65) and substituting the value of coefficients C_i^{k+1} into the solution structure (2.40) yields an approximate solution T^{n+1} to the quasi-steady nonlinear problem at time t_{n+1}.

2.6 Solving the Nonlinear Transient Problem

The numerical algorithms involve stepping through time by step size Δt. At each time step equations (2.20), (2.30, 2.54 and 2.64 are solved by an iterative algorithm, and the superscripts $k + 1$ and k in the equations denote solutions computed at the current and previous iterations respectively. The iterative process is stopped as soon as the difference between two consecutive solutions becomes acceptably small.

At the first time step, we begin the iterative procedure by assuming a constant value for temperature-dependent $\lambda(T)$ term and solving the problem as steady-state. At the next iteration $k + 1$, $\lambda(T)$ is updated with values evaluated using the solutions obtained at the previous iteration k. This updating continues until the difference between two consecutive solutions becomes sufficiently small. The difference between two consecutive solutions is measured by computing the difference as ratio $\delta = \sum_{i=1}^{K}(C_i^{k+1} - C_i^k)^2 / \sum_{i=1}^{K}(C_i^{k+1})^2$. Substituting the value of the coefficients C_i^{k+1} into the solution structure yields an approximate solution to the quasi-steady problem at time t_{n+1}.

At the next time step, we begin the iterative process by updating the $\lambda(T)$ term with values evaluated using the converged solution T^n obtained at the previous time step t_n. Again, the updating of the thermal conductivity $\lambda(T)$ term continues until converged solution is obtained. At this point we solved the quasi-steady problem and the solution in propagated by time step delta Δt.

CHAPTER 3

NUMERICAL EXPERIMENTS

3.1 Construction of Approximate Distance Fields

We saw in Chapter 2 that formulation of a meshfree method with distance fields solution structure depends critically on the ability to construct a distance field ω for the boundary of the geometric domain. The function ω is constructed in such a way that it vanishes precisely on the boundary of the geometric domain and nowhere else. Equation $\omega = 0$ defines the geometry of the domain implicitly, and such functions ω are called implicit functions for the specified geometric domain. In this thesis work, construction of the implicit functions has been solved using the theory of R-functions [21, 23]. The theory of R-functions was developed in Ukraine by Rvachev and his students and is well documented in the Russian language. Documentation of the theory of R-functions is also available in English [20, 22].

An R-function is a real-valued function whose sign is completely determined by the signs of its arguments. For example, the function xyz can be negative only when the number of its arguments is odd. A similar property is possessed by functions $x+y+\sqrt{xy+x^2+y^2}$ and $xy+z+|z-yx|$, and so on. Such functions encoded Boolean logic functions and called R-functions. Every Boolean function is a companion to infinitely many R-function, which form a branch of set of R-functions. For example, it is well known that $\min(x_1, x_2)$ is an R-function whose companion Boolean function is logical "and" ($\wedge$), and $\max(x_1, x_2)$ is an R-function whose companion Boolean function is logical "or" ($\vee$). But the same branches of R-functions contain many other functions, example:

$$\begin{aligned} x_1 \wedge_\alpha x_2 &\equiv \frac{1}{1+\alpha}\left(x_1 + x_2 - \sqrt{x_1^2 + x_2^2 - 2\alpha x_1 x_2}\right); \\ x_1 \wedge_\alpha x_2 &\equiv \frac{1}{1+\alpha}\left(x_1 + x_2 + \sqrt{x_1^2 + x_2^2 - 2\alpha x_1 x_2}\right), \end{aligned} \tag{3.1}$$

where $\alpha(x_1, x_2)$ is an arbitrary function such that $-1 < \alpha(x_1, x_2) \leq 1$. The precise value of α may not matter, and often it can be set to a constant. For example setting $\alpha = 1$ yields the functions min and max respectively, but setting $\alpha = 0$ results in function $\vee_0$ and $\wedge_0$ [17], that are analytic everywhere except when $x_1 = x_2 = 0$.

Using R-functions, any object defined by a predicate on geometric domain can be represented by a single function ω. The function ω can be evaluated, differentiated, and possesses many other useful properties such as:

- function ω can be constructed in a logical fashion and can be controlled through intuitive user-defined parameters;
- ω can be normalized, in which case it behaves as a distance function near the boundary of the object and can be differentiated everywhere [20];
- the function can also be constructed for individual cells and cells complex, given prescribed values for the function and their gradients.

The theory of R-functions provides the connection between logical and set operations on geometric primitives and analytic constructions. For every logical or set-theoretic construction, there is a corresponding approximate distance function with the above properties. Furthermore, the translation from logical and set-theoretic description is a matter of simple syntactic substitution that does not require expensive symbolic computations. For example, the geometric domain in Figure 3.1(a) can be defined as a Boolean (Constructive Solid Geometry) of two primitives:

$$\Omega = \omega_1 \cap \omega_2,$$

where the individual primitives ω_1 Figure 3.1(b) and ω_2 Figure 3.1(c) are defined by the following inequalities:

$$\omega_1 = 1 - y^2 - x^2 \geq 0; \qquad \omega_2 = -0.0625 + y^2 + x^2 \geq 0.$$

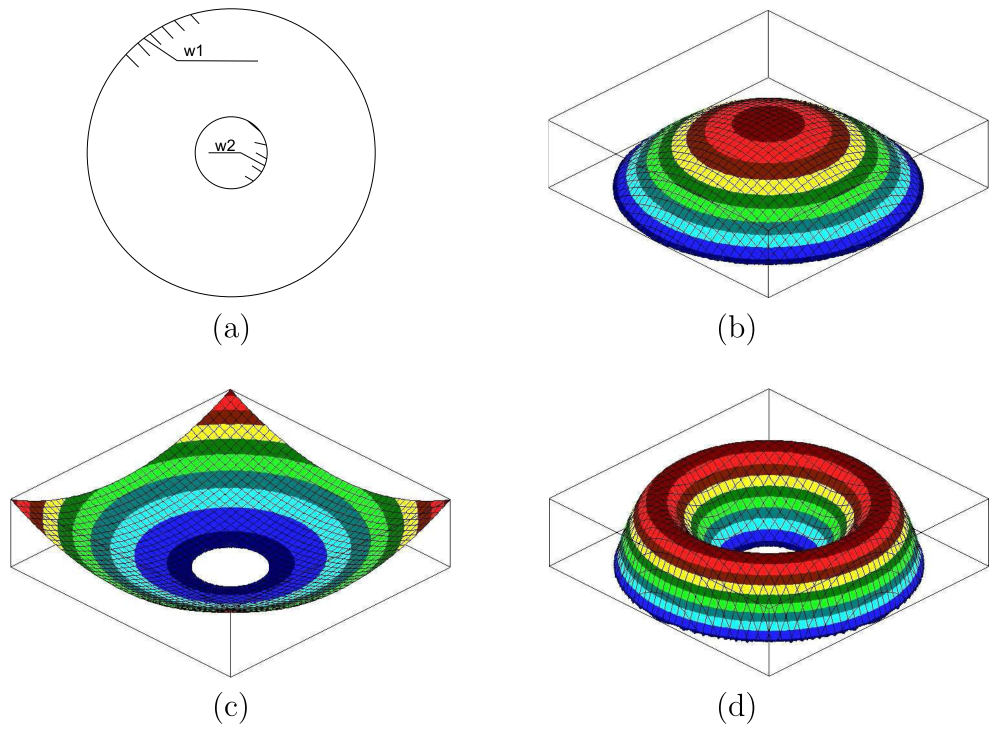


Figure 3.1: (a) Two dimensional representation of the geometric domain of the benchmark problem; (b) outer boundary representation ω_2; (c) inner boundary representation ω_1; (d) the corresponding approximate distance field.

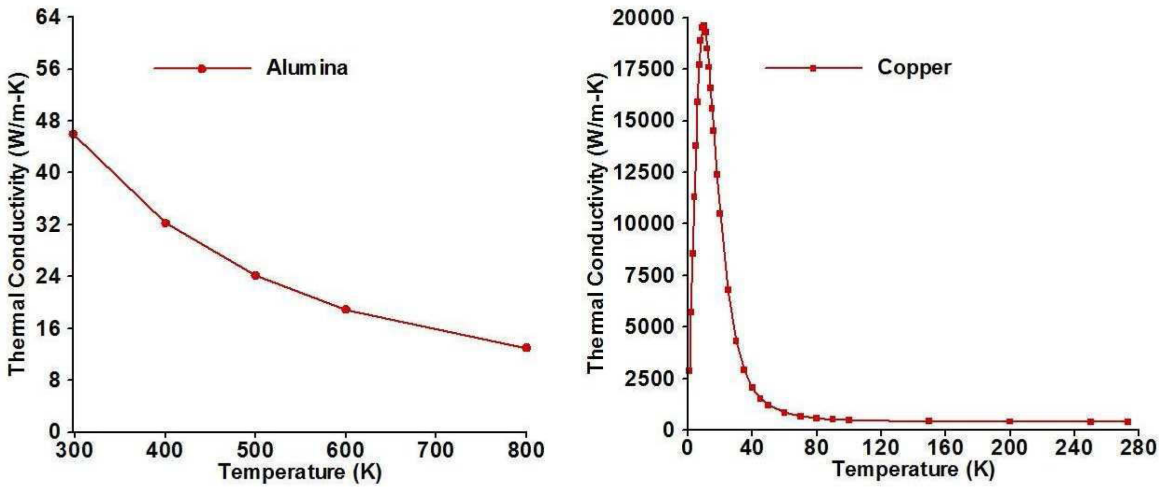
The constructed Boolean representation can be translated into the approximate distance field shown in Figure 3.1(d) using R--functions:

$$\omega = \omega_1 \wedge_0 \omega_2. \tag{3.2}$$

This example clearly shows any Boolean representation of a geometric domain may be translated into the corresponding approximated distance field. This logical description can also be directly translated into a function such that is zero for every point on the boundary and positive elsewhere.

3.2 Computation Validation

The developed algorithms for meshfree method with distance fields solution procedure has been validated on the simple two-dimensional benchmark problem shown in Figure 3.1(a). Accuracy and convergence of the numerical solutions computed by the meshfree method with distance fields are compared to analytical solutions, and solutions produced by commercial FEM software ANSYS 12.1. We conducted numerical experiments for steady-state and transient heat transfer problems with constant and temperature-dependent thermal conductivities.



(a) (b)

Figure 3.2: (a)Temperature-dependency of thermal conductivity for alumina. Data taken from online materials database matweb. (b) Temperature-dependency of thermal conductivity for copper. Data taken from Journal of Physical and Chemical Reference Data.

We studied the behavior of our developed meshfree method with distance fields algorithms to solve problems for a material with weak temperature-dependency of thermal conductivity (alumina), and for a material with very steep temperature-dependency of thermal conductivity (copper).

Copper has very steep temperature-dependency of thermal conductivity in range of very low temperatures. At very low temperature, thermal conductivity of copper reaches very high values because the lattice waves are harmonic and can be superimposed without mutual interference [12]. There, the lattice thermal conductivity of crystals depends upon the grain size. As the temperature increases, the lattice vibrations become nonharmonic, scattering is increased, and the thermal conductivity decreases sharply. Figure 3.2(b) shows the actual dependence of the thermal conductivity of copper on the temperature. Alumina has a weaker temperature-dependency of thermal conductivity as shown in Figure 3.2(a).

3.3 Experiment 1: Steady-State Problem with Constant Thermal Conductivity and Dirichlet Boundary Conditions

The meshfree method with distance fields was applied to modeling steady-state heat transfer in the benchmark problem Figure 3.1(a) with Dirichlet boundary conditions. The Dirichlet boundary conditions are formulated by a prescribed temperature of 1 °K on the outer boundary ω_1, and a prescribed temperature of 40 °K on the inner boundary ω_2. The temperature distribution is represented by the solution structure (2.13). Basis functions $\{\chi_i\}_{i=1}^{N}$ in the solution structure have been chosen as B-splines of the fourth degree, distributed over a uniform rectangular grid.

Analytical solution of this problem is available from a very easily solvable equation of the form

$$T = C_1 ln(r) + C_2, \tag{3.3}$$

where

$$r = \sqrt{x^2 + y^2} \tag{3.4}$$

and

$$\begin{aligned} C_1 ln(0.25) + C_2 &= 40 \\ C_2 ln(1) + C_2 &= 1. \end{aligned} \tag{3.5}$$

Meshfree method with distance fields allows the given boundary conditions to be satisfied exactly and it discretizes not the geometric domain but the underlying functional space. However, meshfree does not necessarily imply the absence of spacial grid. A grid may be convenient or even necessary for integration and/or visualization purposes. In this experiment, we studied the error of the meshfree method with distance fields approximation. In particular, we studied the error that is introduced in the numerical computations due to grid size specification.

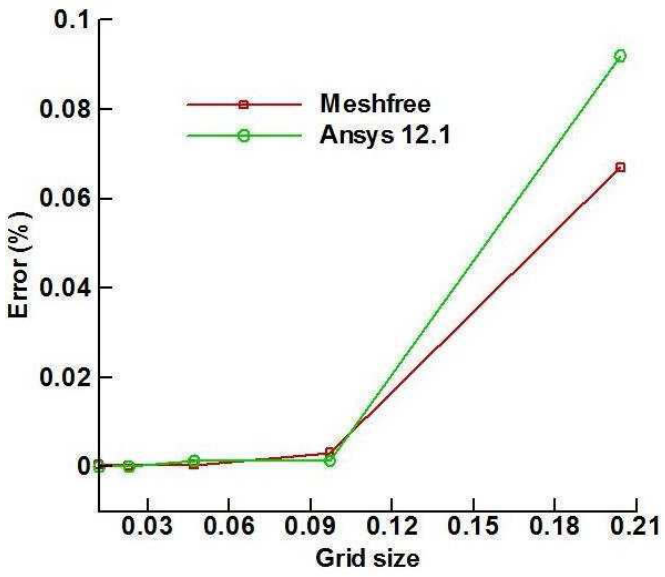


Figure 3.3: Convergence in terms of estimated errors produced by meshfree method with distance fields and FEM software ANSYS 12.1 for five different grid sizes.

To study this error of the meshfree method with distance fields we solved the problem using five different uniform rectangular grids: 11×11, 22×22, 44×44, 88×88, and 176×176 grids of B-splines. The error of the meshfree method with distance fields is estimated by the L2-norm of the form (3.6), computed over the domain Ω.

$$\varepsilon_{meshfree} = \frac{||T_{meshfree} - T_{exact}||}{||T_{exact}||} = \frac{\sqrt{\int_{\Omega}(T_{meshfree} - T_{exact})^2 \, d\Omega}}{\sqrt{\int_{\Omega}(T_{exact})^2 \, d\Omega}} \tag{3.6}$$

We compared the meshfree method with distance fields error with error produced by commercial FEM software ANSYS 12.1 for the same grid size specifications that we used for meshfree method with distance fields. Here, grid size means the mesh size or element size h, which is calculated from (3.7)

$$h = \frac{x_{max} - x_{min}}{n_x - 1}, \tag{3.7}$$

where x_{max} and x_{min} are the maximum and minimum size of the bounding box used for meshfree method with distance fields grid, and n_x is the number of grids.

The error produced by the ANSYS 12.1 is estimated by computing the difference (L2-norm) between the "exact" temperature and temperature predicted by ANSYS 12.1 at location $x = 0.8, y = 0.0$ in the geometrical domain of the benchmark problem.

$$\varepsilon_{FEM} = \frac{\sqrt{(T_{FEM} - T_{exact})^2}}{\sqrt{(T_{exact})^2}} \tag{3.8}$$

The plots shown in Figure (3.3) clearly illustrates that the meshfree method with distance fields converged smoothly to the "exact" solution. We estimated that the error produced by meshfree method with distance fields and ANSYS 12.1 becomes sufficiently small for a 60×60 rectangular grid size ($h = 0.034$). Therefore, for all subsequent numerical experiments conducted on the benchmark problem, we studied the behavior of the meshfree method with distance fields algorithms using a unform 60×60 rectangular grid of B-splines, and an equivalent element size $h = 0.034$ for ANSYS 12.1.

Table 3.1: The relative difference between meshfree method with distance fields and analytical solutions: steady-state problem with Dirichlet boundary conditions and constant thermal conductivity.

Location (x, y)	Analytical Solution (K)	Meshfree Solution (K)	Relative Difference between Analytical & Meshfree (%)
0.4, 0	26.777598	26.776998	0.0022
0.5, 0	20.5	20.499527	0.0023
0.6, 0	15.370829	15.370474	0.0023
0.7, 0	11.034177	11.033921	0.0023
0.8, 0	7.277598	7.277434	0.0022

Accuracy of the meshfree method with distance fields computation is confirmed by the experimental results shown in Table 3.1. The results show that temperatures computed by meshfree method with distance fields at five different locations in the geometric domain of the benchmark problem, are essentially identical to temperatures

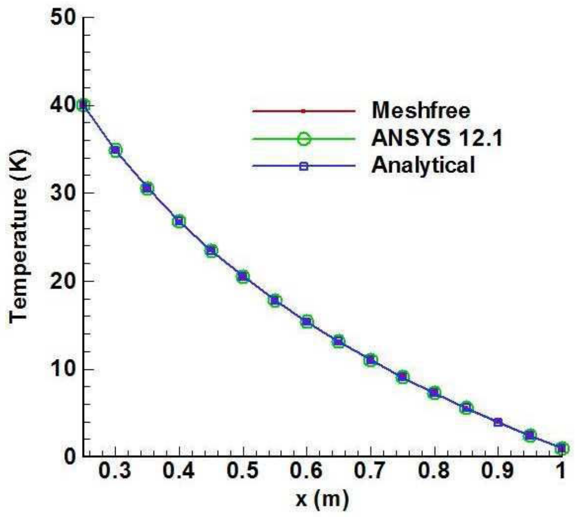


Figure 3.4: Temperature distribution in the benchmark problem Figure (3.1a) computed analytically, by meshfree method with distance fields and ANSYS 12.1: steady-state problem with Dirichlet boundary conditions and constant thermal conductivity.

calculated analytically. The meshfree method with distance fields solution is further validated by the temperature distribution shown in Figure 3.4. Figure 3.4 demonstrates that the temperature distribution computed by meshfree method with distance fields is almost identical to the temperature distribution computed analytically and by ANSYS 12.1.

3.4 Experiment 2: Steady-State Problem with Constant Thermal Conductivity and Convective Boundary Conditions

The meshfree method with distance fields was applied to modeling steady-state heat transfer in the benchmark problem Figure 3.1(a) with convective boundary conditions. The convective boundary conditions are formulated by setting the temperature of the medium surrounding the outer boundary ω_1 to 1 °K, and setting the temperature of the medium surrounding the inner boundary ω_2 to 40 °K. Heat transfer on the boundaries is at 450 W/m^2-K. The temperature distribution is represented by the solution structure (2.38). Basis functions $\{\chi_i\}_{i=1}^{N}$ in the solution structure have been chosen as B-splines of the fourth degree, distributed over a uniform rectangular grid. The solution

Table 3.2: The relative difference between meshfree method with distance fields and ANSYS 12.1 solutions: steady-state problem with convective boundary conditions and constant thermal conductivity.

Location (x, y)	Meshfree Solution (K)	ANSYS 12.1 Solution (K)	Relative Difference between Meshfree & ANSYS 12.1 (%)
0.4, 0	20.9066	20.907	0.0019
0.5, 0	16.679	16.679	0.0000
0.6, 0	13.2248	13.305	0.0015
0.7, 0	10.3044	10.305	0.0058
0.8, 0	7.7746	7.7748	0.0026

computed by meshfree method with distance fields is compared to solution produced by ANSYS 12.1.

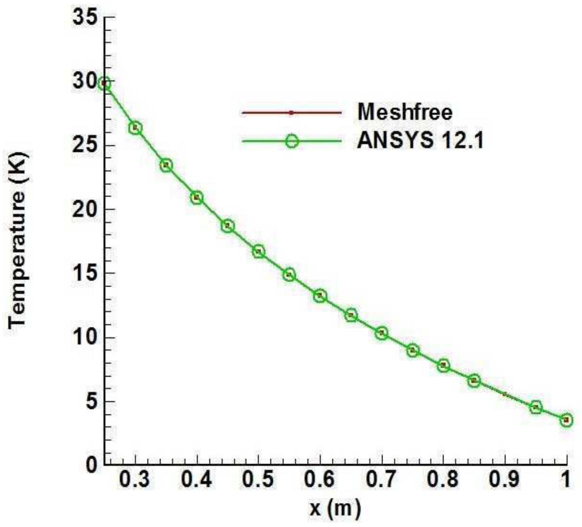


Figure 3.5: Temperature distribution in the benchmark problem Figure (3.1a) computed by meshfree method with distance fields and ANSYS 12.1: steady-state with convective boundary conditions and constant thermal conductivity.

Accuracy of the numerical computations by computed by meshfree method with distance fields is confirmed by the experimental results shown in Table 3.2. The results illustrate that temperatures computed by meshfree method with distance fields and temperatures predicted by FEM software ANSYS 12.1 at five different locations in the benchmark problem, exhibits almost no difference. The meshfree method with distance fields solution is further validated by the temperature distributions presented in Figure

3.5. The plots in Figure 3.5 illustrates that the temperature distributions computed by meshfree method with distance fields is almost identical to the temperature distributions produced by ANSYS 12.1.

3.5 Experiment 3: Transient Problem with Constant Thermal Conductivity and Dirichlet Boundary Conditions

The meshfree method with distance fields was applied to modeling transient heat transfer in the benchmark problem Figure 3.1(a) with Dirichlet boundary conditions. The Dirichlet boundary conditions are formulated by a prescribed temperature of 1 °K on the outer boundary ω_1, and a prescribed temperature of 40 °K on the inner boundary ω_2. The temperature distribution is represented by the solution structure (2.13). Basis functions $\{\chi_i\}_{i=1}^{N}$ in the solution structure have been chosen as B-splines of the

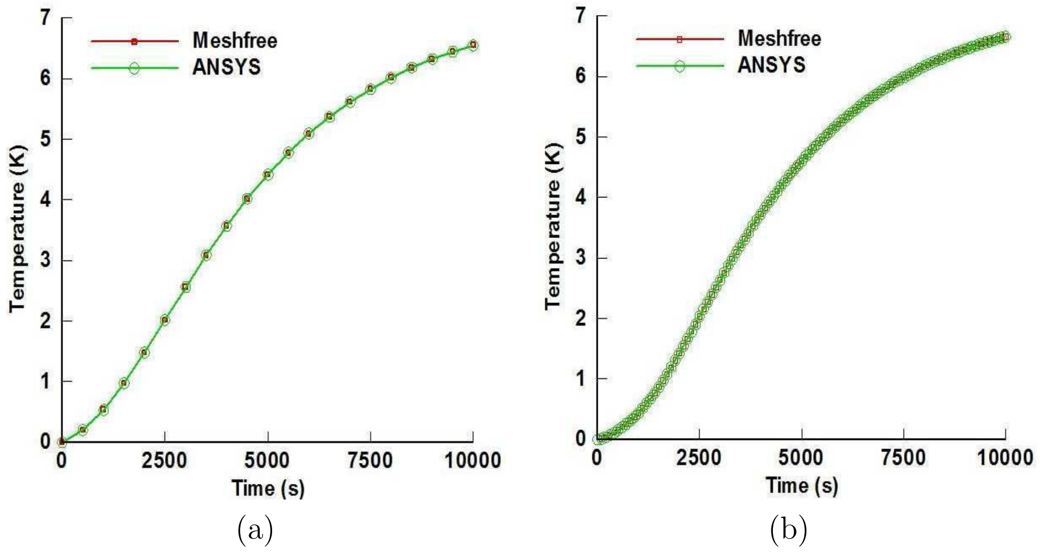
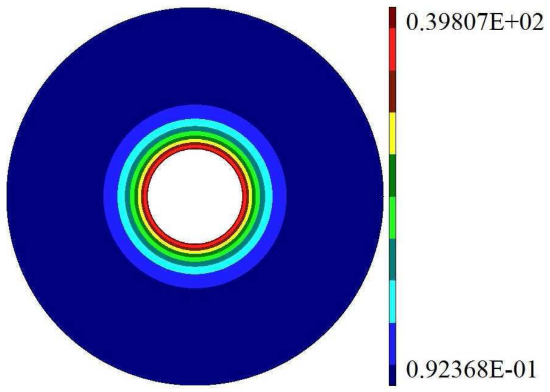
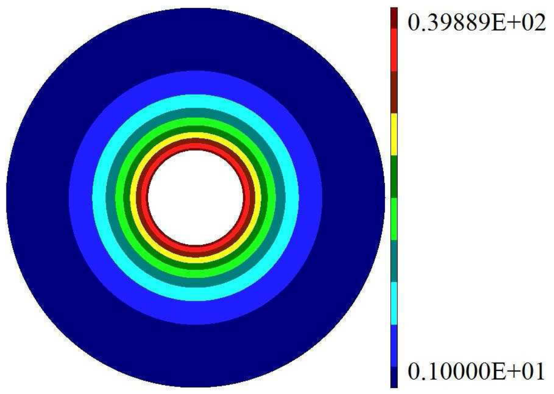


Figure 3.6: Comparing time evolution of temperature at location $x = 0.8, y = 0.0$ predicted by meshfree method with distance fields and ANSYS 12.1: for two different time steps, constant thermal conductivity, and Dirichlet boundary conditions.

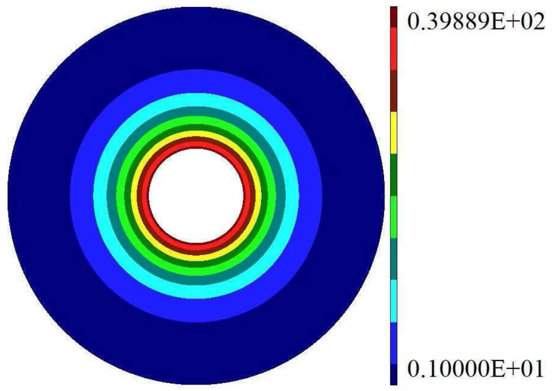
Figure 3.6 shows temperature prediction at location $x = 0.8, y = 0.0$ in the benchmark problem with thermal conductivity $\lambda = 60.5$ W/m-K, density $\rho = 7850$ kg/m^3 and specific heat $c = 434$ J/kg-K. The problem is defined to have an initial uniform temperature of $T = 0.0$ °K.



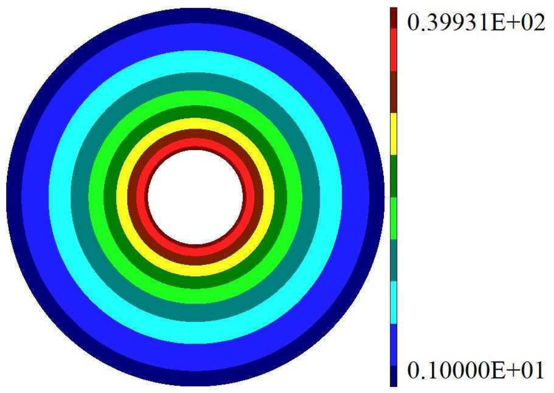
(a) 500 sec



(b) 2000 sec.



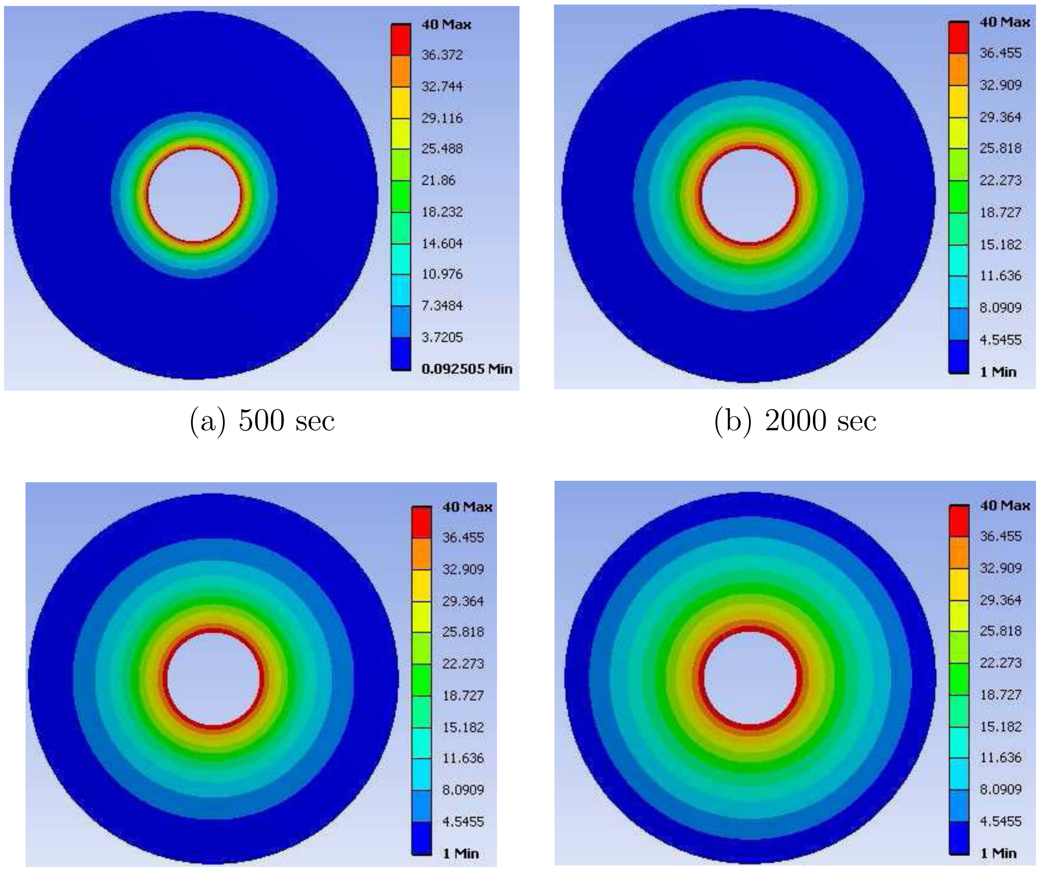
(c) 4000 sec



(d) 10000 sec

Figure 3.7: Quasi-steady temperature fields computed by meshfree method with distance fields: constant thermal conductivity, Dirichlet boundary conditions, and time step $t = 100$ sec

Figures 3.6(a) and (b) show time evolution of the temperature at location $x = 0.8, y = 0.0$, computed by meshfree method with distance fields and ANSYS 12.1 for two different time steps: 500 seconds in Figure 3.6(a) and 100 seconds in Figure 3.6(b). The plots illustrate that time evolution of the temperature computed by meshfree method with distance fields is essentially identical that of ANSYS 12.1. Comparison of Figure 3.7 with Figure 3.8 shows almost no difference between the quasi-steady temperature fields computed by meshfree method with distances and quasi-steady temperature fields produced by ANSYS 12.1.



(a) 500 sec

(b) 2000 sec

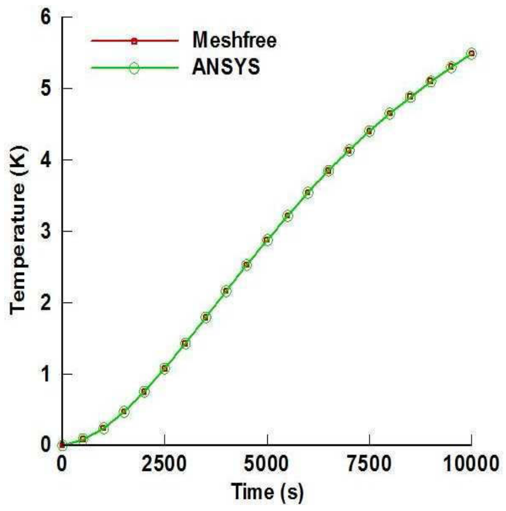
(c) 4000 sec

(d) 10000 sec

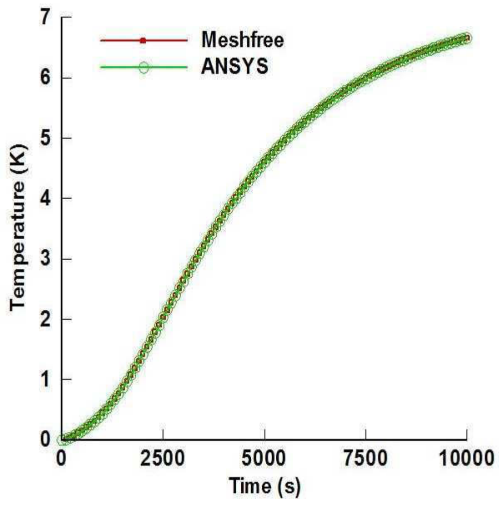
Figure 3.8: Quasi-steady temperature fields computed by ANSYS 12.1: constant thermal conductivity, Dirichlet boundary conditions, and time step $t = 100$ sec.

3.6 Experiment 4: Transient Problem with Constant Thermal Conductivity and Convective Boundary Conditions

The meshfree method with distance fields was applied to modeling transient heat transfer in the benchmark problem Figure 3.1(a) with convective boundary conditions. The convective boundary conditions are formulated by setting the temperature of the medium surrounding the outer boundary ω_1 to 1 °K, and setting the temperature of the medium surrounding the inner boundary ω_2 to 40 °K. Heat transfer on the boundaries is at 450 W/m^2-K. The temperature distribution is represented by the solution structure (2.38). Basis functions $\{\chi_i\}_{i=1}^N$ in the solution structure have been chosen as B-splines of the fourth degree, distributed over a uniform rectangular grid. The solution computed by meshfree method with distance fields is compared to solution produced by ANSYS 12.1 using the same grid size specification. Figure 3.9 shows temperature prediction



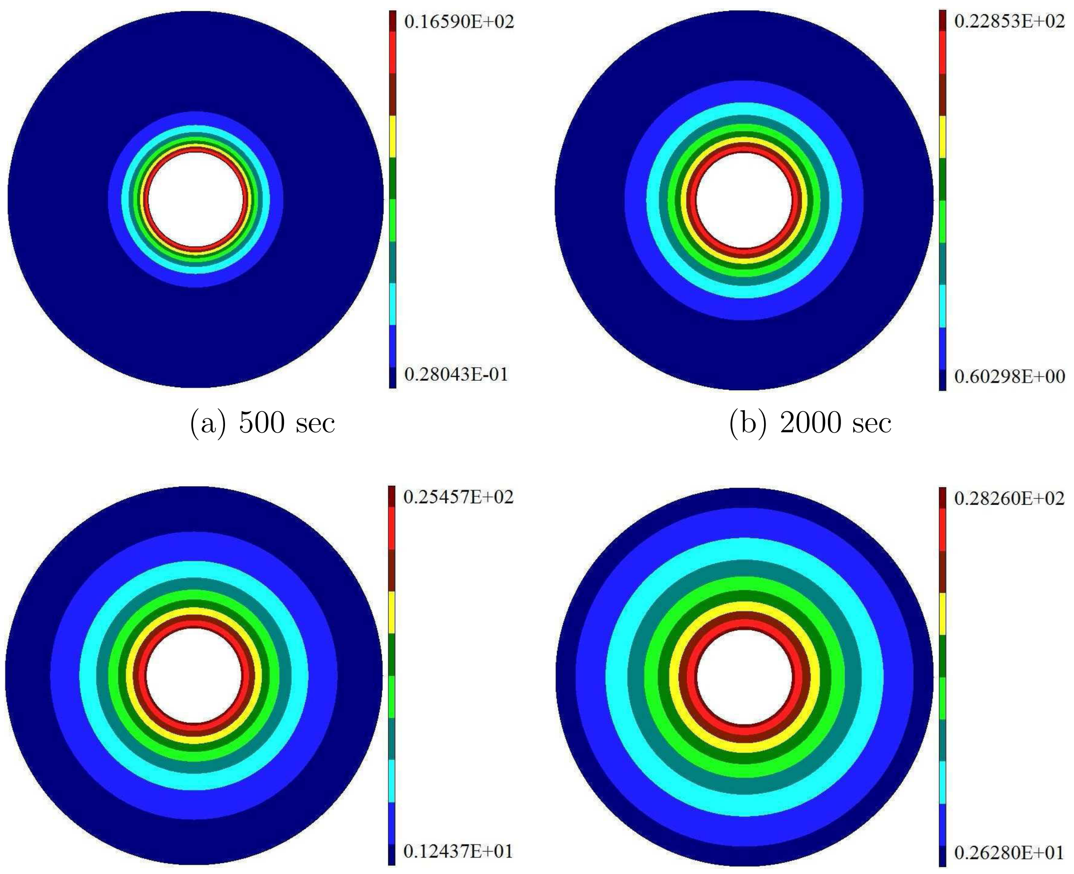
(a)



(b)

Figure 3.9: Comparing time evolution of the temperature at location $x = 0.8, y = 0.0$ predicted by meshfree method with distance fields and ANSYS 12.1: for two different time steps, constant thermal conductivity, and convective boundary conditions.

at location $x = 0.8, y = 0.0$ in the benchmark problem with thermal conductivity $\lambda =$ 60.5 W/m-K, density $\rho =$ 7850 kg/m3 and specific heat $c =$ 434 J/kg-K. The problem is defined to have an initial uniform temperature of $T = 0.0$ °K. Figures 3.9(a) and (b) show the time evolution of the temperature at location $x = 0.8, y = 0.0$, computed by meshfree method with distance fields and by FEM software ANSYS 12.1 for two different time steps: 500 seconds in Figure 3.9(a) and 100 seconds in Figure 3.9(b). The plots illustrate that time evolution of the temperature computed by meshfree method with distance fields is essentially identical that of ANSYS 12.1. Comparison of Figure 3.10 with Figure 3.11 shows almost no difference between the quasi-steady temperature fields computed by meshfree method with distances and ANSYS 12.1.



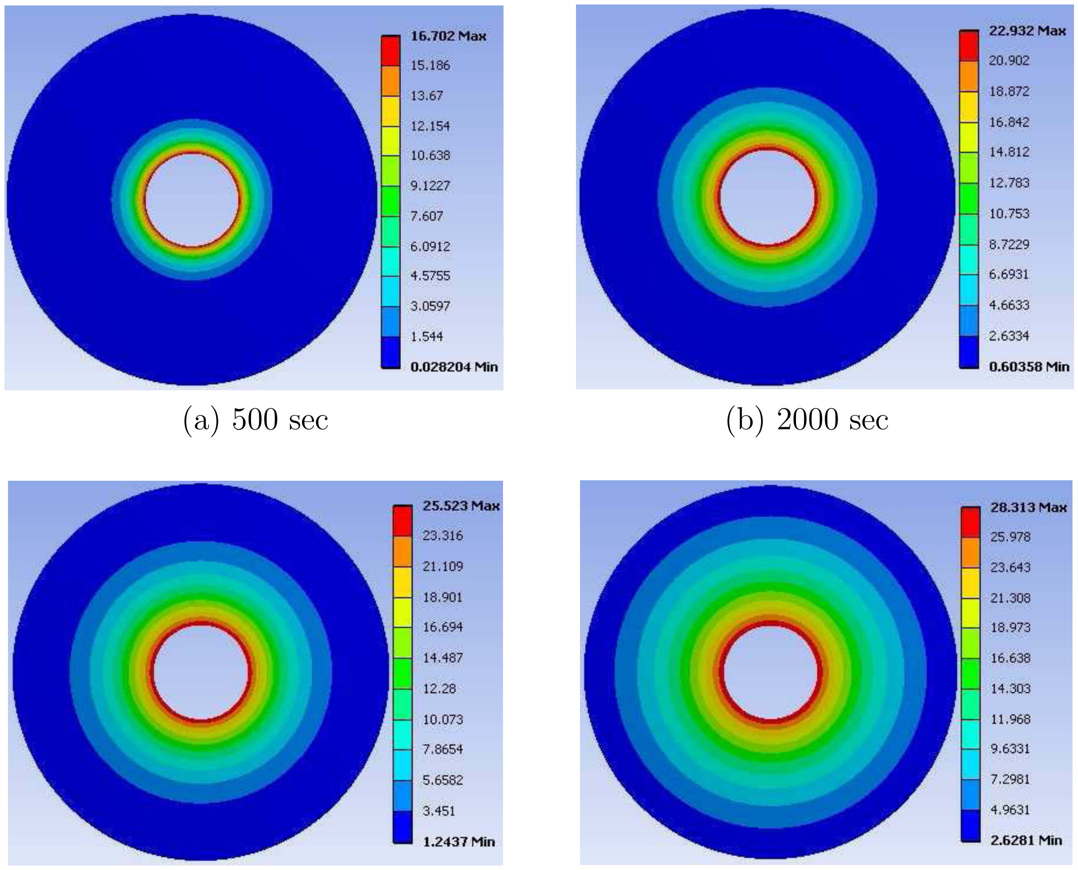
(a) 500 sec (b) 2000 sec

(c) 4000 sec (d) 10000 sec

Figure 3.10: Quasi-steady temperature fields computed by meshfree method with distance fields: constant thermal conductivity, convective boundary conditions, and time step $t = 100$ sec.

3.7 Nonlinear Heat Transfer Problems with Dirichlet Boundary Conditions

The meshfree method with distance fields was applied to modeling nonlinear steady-state and transient heat transfer in the benchmark problem (Figure 3.1(a)) with Dirichlet boundary conditions. The temperature distribution is represented by the solution structure (2.13). Basis functions $\{\chi_i\}_{i=1}^{N}$ in the solution structure have been chosen as B-splines of the fourth degree distributed over a uniform rectangular grid. To validate the accuracy and convergence properties of the meshfree method with distance fields we compared solutions computed by meshfree method with distance with ANSYS 12.1 solutions.



(a) 500 sec (b) 2000 sec

(c) 4000 sec (d) 1000 sec

Figure 3.11: Quasi-steady temperature fields computed by ANSYS 12.1: constant thermal conductivity, convective boundary conditions, and time step $t = 100$ sec.

Solutions were computed by meshfree method with distance fields using algorithms based on Oseen linearization scheme and Newton-Kantorovich linearization scheme. The linearization schemes applied to linearize the nonlinear terms in the equations that defined the problem leads to an iterative procedure. The convergence of iterative solutions is achieved when the difference between two consecutive solutions becomes sufficiently small. In this experiment, we measured the relative difference between two consecutive solutions by the ratio

$$\varepsilon = \frac{\sum_{i=1}^{K}(C_i^{k+1} - C_i^k)^2}{\sum_{i=1}^{K}(C_i^{k+1})^2}, \tag{3.9}$$

where C_i^{k+1} and C_i^k are the values of the coefficients computed at the current and previous iterations respectively. The iterations are stopped as soon as $\varepsilon \leq 10^{-6}$. For both steady-state and transient heat transfer problems with Dirichlet boundary conditions, we conducted experiments using a material with weak temperature-dependency of thermal conductivity, and a material with very steep temperature-dependency of thermal conductivity.

3.7.1 Experiment 5: Steady-state problem with weak temperature dependency of thermal conductivity

We applied the meshfree method with distance fields to modeling heat transfer in the benchmark problem Figure 3.1(a) made from a material with weak temperature-dependency of thermal conductivity (alumina).

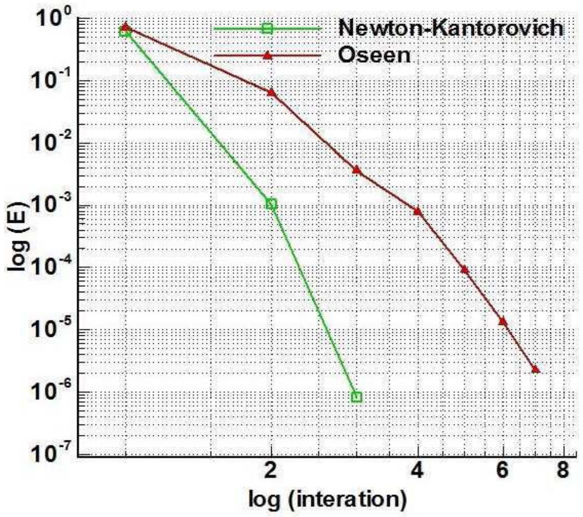


Figure 3.12: Convergence of Oseen and Newton-Kantorovich linearization schemes in terms of relative difference between two consecutive solutions ε (Equation 3.9): observed for a material with weak temperature-dependence of thermal conductivity.

The temperature-dependency of thermal conductivity data for alumina, used in this experiment, is taken from Figure 3.2(a). We observed the behavior of the meshfree method with distance fields using algorithms based on Oseen and Newton-Kantorovich linearization schemes. The Dirichlet boundary conditions are formulated by a prescribed temperature of 320 °K on the outer boundary ω_1, and a prescribed temperature of 650 °K on the inner boundary ω_2. The plots shown in Figure 3.12 illustrate that Newton-

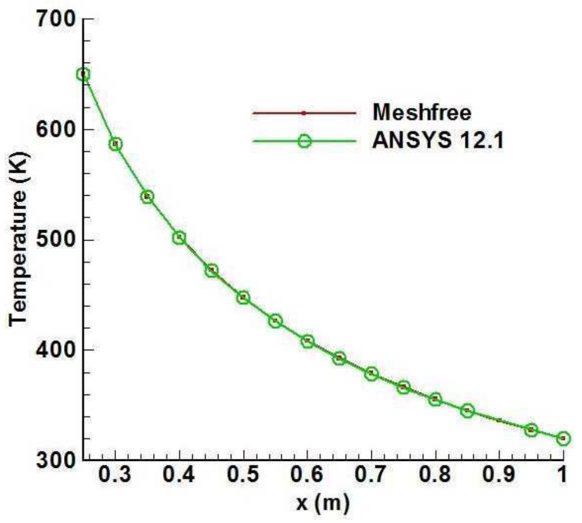


Figure 3.13: Temperature distribution in the benchmark problem Figure (3.1a) computed by meshfree method with distance fields and ANSYS 12.1: steady-state problem with Dirichlet boundary conditions and weak temperature-dependency of thermal conductivity.

Kantorovich linearization scheme produced fast converging solutions in comparison to Oseen linearization scheme, when applied to solve the steady-state problem with Dirichlet boundary, and temperature-dependency of thermal conductivity is weak. Accuracy of the solution computed by meshfree method with distance fields is confirmed by the experimental results shown in Table 3.3. The results illustrate that temperatures computed by meshfree method with distance fields and temperatures predicted by FEM ANSYS

Table 3.3: The relative difference between meshfree method with distance fields and ANSYS 12.1 solutions: steady-state problem with Dirichlet boundary conditions and weak temperature-dependence of thermal conductivity.

Location (x, y)	Meshfree Oseen (K)	Meshfree Newton-Kantorovich (K)	ANSYS 12.1 (K)	Relative Difference Oseen-Ansys (%)	Relative Difference Kantorovich-Ansys (%)
0.4, 0	502.184530	502.184529	502.05	0.0268	0.0268
0.5, 0	447.633637	447.633633	447.4	0.0522	0.0522
0.6, 0	408.637386	408.637386	408.4	0.0581	0.0581
0.7, 0	378.996733	378.996729	378.62	0.0994	0.0994
0.8, 0	355.474403	355.474401	355.27	0.0575	0.0575

12.1, exhibits almost no difference. Comparison of Figures 3.13(a) and (b) with Figure 3.15(c) demonstrate that the temperature distribution computed by meshfree method with distances fields is essentially identical to the temperature distributions produced by ANSYS 12.1.

3.7.2 Experiment 6: Steady-state problem with strong temperature-dependency of thermal conductivity

The meshfree method with distance fields was applied to modeling heat transfer in the benchmark problem Figure 3.1(a) made from a material with very strong temperature-dependency of thermal conductivity (copper). The temperature-dependency of thermal conductivity data for copper,

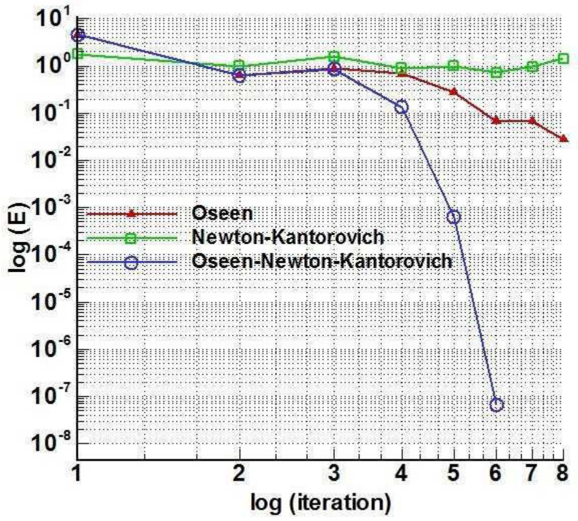


Figure 3.14: Convergence in terms of relative difference between two consecutive solution ε (Equation 3.9). Convergence was observed for Oseen, Newton-Kantorovich, and both Oseen and Newton-Kantorovich applied in the linearization process for a material with very strong temperature-dependence of thermal conductivity.

used in this experiment, is taken from Figure 3.2(b). The Dirichlet boundary conditions are formulated by a prescribed temperature of 1 °K on the outer boundary ω_1, and a prescribed temperature of 60 °K on the inner boundary ω_2. We observed the behavior of the meshfree method with distance fields using algorithms based Oseen and Newton-Kantorovich linearization schemes.

Table 3.4: The relative difference between meshfree method with distance fields and ANSYS 12.1 solutions: Solutions obtained for steady-state with Dirichlet boundary conditions and very strong temperature-dependence of thermal conductivity.

Location (x, y)	Oseen Meshfree (K)	Newton-Kantorovich Meshfree (K)	ANSYS 12.1 (K)	Relative Difference Oseen-Ansys (%)	Relative Difference Kantorovich-Ansys (%)
0.4, 0	18.993867	18.993783	18.76	0.1783	0.1779
0.5, 0	14.305141	14.305152	14.276	0.1947	0.2038
0.6, 0	11.318277	11.318317	11.3	0.1654	0.1618
0.7, 0	9.019745	9.01979	9.0057	0.1181	0.1562
0.8, 0	6.95337	6.953409	6.9415	0.1707	0.1713

We observed that for this defined problem, steady-state with Dirichlet boundary conditions and very strongtemperature-dependency of thermal conductivity, the solution diverges as shown in the plots Figure 3.14 when Newton-Kantorovich linearization scheme is applied. In contrast, the solution is very slow to converged when Oseen linearization scheme is applied. Therefore, we developed numerical algorithm that applied Oseen linearization at the initial iteration, and then applied Newton-Kantorovich when the difference between two consecutive solutions becomes less than one ($\varepsilon > 1$). We observed that by applying both schemes during the linearization process, insure fast converging solution as shown in Figure 3.14 if the temperature-dependency of thermal conductivity is very strong.

Accuracy of the computation by meshfree method with distance fields is confirmed by the experimental results shown in Table 3.4. The results illustrate that temperatures computed by meshfree method with distance fields and temperatures predicted by FEM ANSYS 12.1 exhibits almost no difference. Comparison of Figure 3.15(a) with Figure 3.15(b) demonstrate that the temperature distribution computed by meshfree method with distances fields is essentially identical to the temperature distributions produced by ANSYS 12.1.

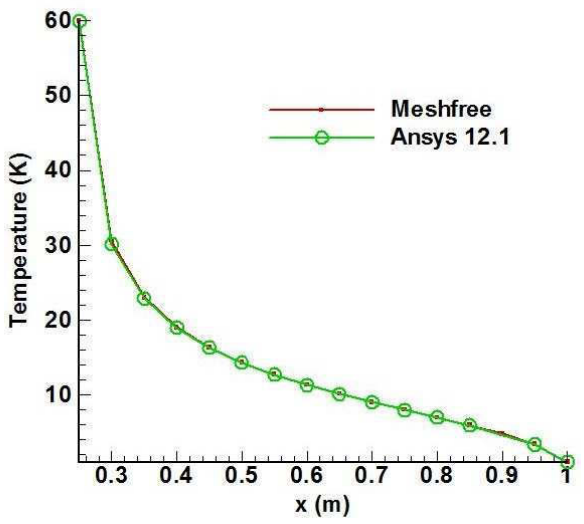
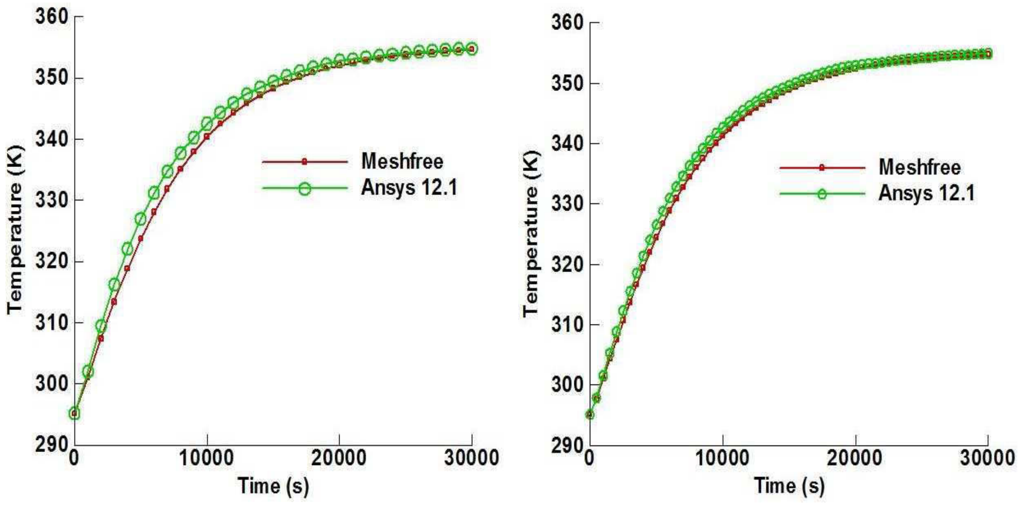


Figure 3.15: Temperature distribution in the benchmark problem Figure (3.1a) computed by meshfree method with distance fields and ANSYS 12.1: steady-state problem with Dirichlet boundary conditions and very strong temperature-dependency of thermal conductivity.

3.7.3 Experiment 7: Transient problem with weak temperature dependency of thermal conductivity

The Dirichlet boundary conditions are formulated by a prescribed temperature of 320 °K on the outer boundary ω_1, and a prescribed temperature of 650 °K on the inner boundary ω_2. Figure 3.16 shows temperature prediction at location $x = 0.8, y = 0.0$ in the benchmark problem with weak temperature-dependence of thermal conductivity, $\lambda(T)$ taken from data given in Figure 3.2(a), density $\rho = 3960$ kg/m^3 and specific heat $c = 850$ J/kg-K. The problem is defined to have an initial uniform temperature of T = 215.15 °K. Figures 3.16(a) and (b) show the time evolution of the temperature at location $x = 0.8, y = 0.0$, computed by meshfree method with distance fields

and by FEM software ANSYS 12.1 for two different time steps: 1000 seconds in Figure 3.16(a) and 500 seconds in Figure 3.26(b). The plots shown in Figure 3.16 illustrate that time evolution of the temperature computed by meshfree method with distance fields is essentially identical that of ANSYS 12.1.

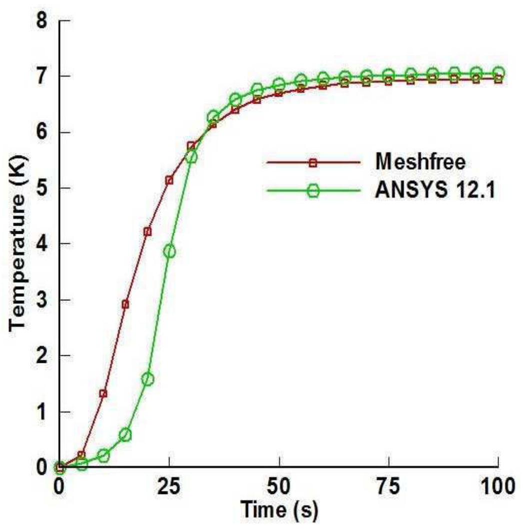


(a) (b)

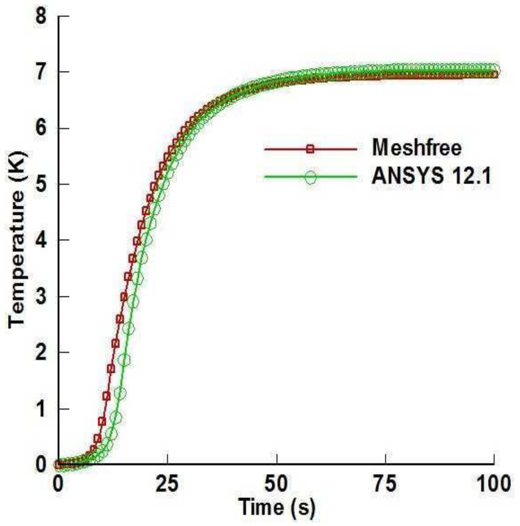
Figure 3.16: Comparing time evolution of the temperature at location $x = 0.8, y = 0.0$ predicted by meshfree method with distance fields and ANSYS 12.1: for two different time steps, Dirichlet boundary conditions and weak temperature-dependence of thermal conductivity.

3.7.4 Experiment 8: Transient problem with strong temperature dependency of thermal conductivity

The Dirichlet boundary conditions are formulated by a prescribed temperature of 1 °K on the outer boundary ω_1, and a prescribed temperature of 60 °K on the inner boundary ω_2. Figure 3.17 shows temperature prediction at location $x = 0.8, y = 0.0$ in the benchmark problem with strong temperature-dependence of thermal conductivity, $\lambda(T)$ taken from data given in Figure 3.2(b), density $\rho = 8300$ kg/m^3 and specific heat $c = 385$ J/kg-K. The problem is defined to have an initial uniform temperature of $T = 0.0$ °K. Figures 3.17(a) and (b) show the time evolution of the temperature at location $x = 0.8, y = 0.0$, computed by meshfree method with distance fields and by FEM software ANSYS 12.1 for two different time steps: 5 seconds in Figure 3.17(a) and 1 seconds in Figure 3.17(b).



(a)



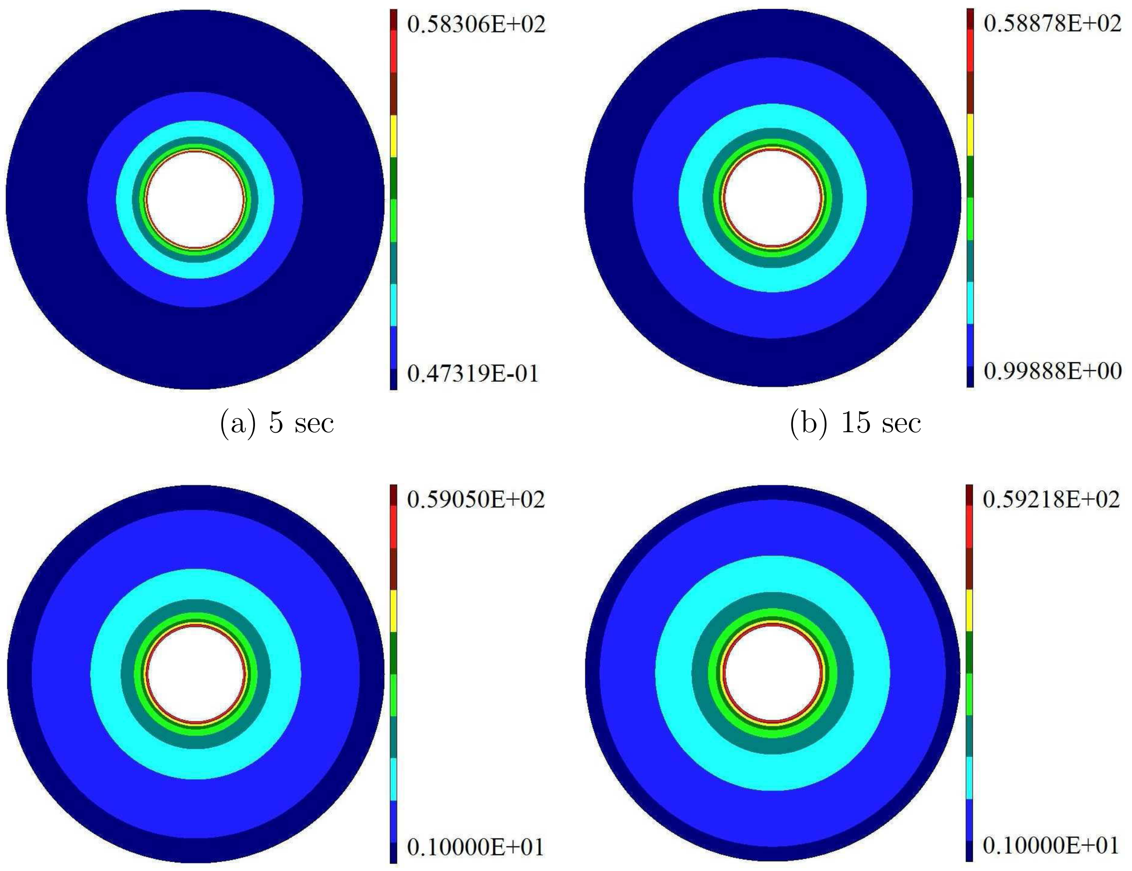
(b)

Figure 3.17: Comparing time evolution of the temperature at location $x = 0.8, y = 0.0$ predicted by meshfree method with distance fields and ANSYS 12.1: for two different time steps, Dirichlet boundary conditions and strong temperature-dependence of thermal conductivity.

The plots shown in Figure 3.17 illustrate that time evolution of the temperature computed by meshfree method with distance fields is essentially identical that of ANSYS 12.1. Comparison of Figure 3.18 with Figure 3.19 shows almost no difference between the quasi-steady temperature fields computed by meshfree method with distances and the quasi-steady temperature fields predicted by ANSYS 12.1. This experiment also confirm that the accuracy of the meshfree method with distance fields computation to obtain solution of nonlinear transient heat transfer problem using material with strong temperature-dependency of thermal conductivity and Dirichlet boundary conditions.

3.8 Nonlinear Heat Transfer Problems with Convective Boundary Conditions

The meshfree method with distance fields was applied to modeling nonlinear steady-state and transient heat transfer in the benchmark problem Figure 3.1(a) with convective boundary conditions. The temperature distribution is represented by the solution structure (2.38). Basis functions $\{\chi_i\}_{i=1}^{N}$ in the solution structure have been



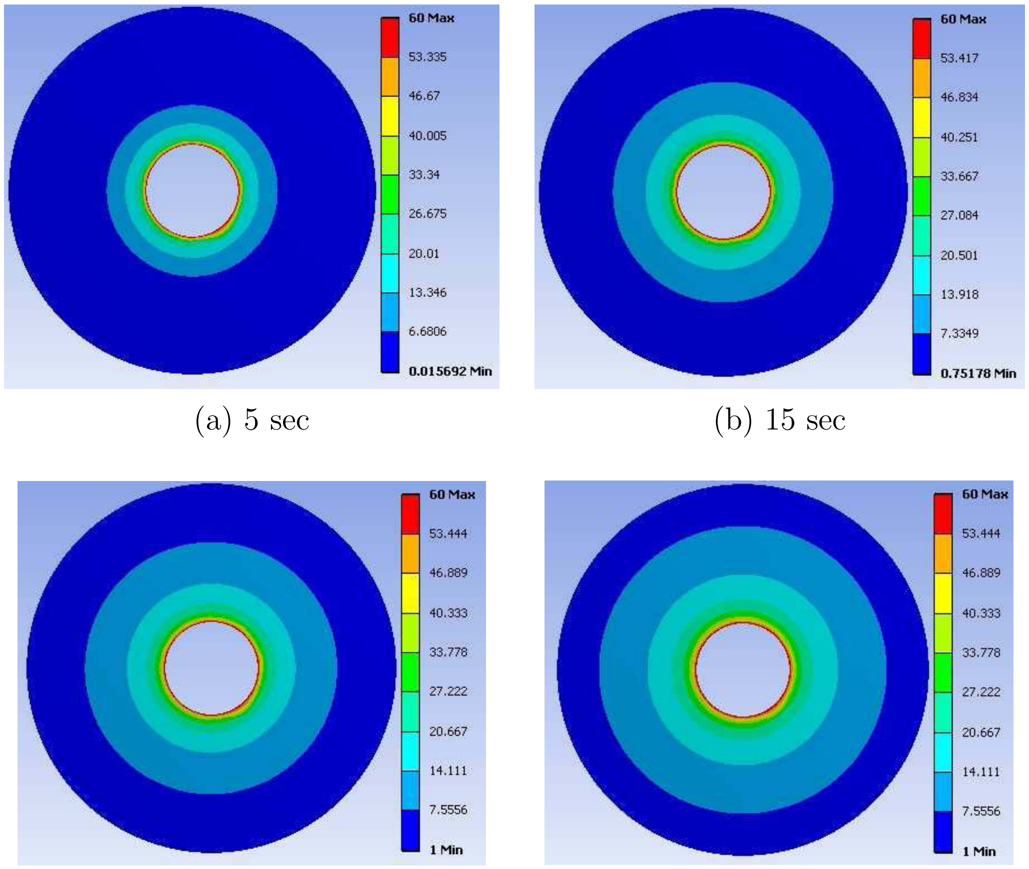
(a) 5 sec (b) 15 sec

(c) 25 sec (d) 100 sec

Figure 3.18: Quasi-steady temperature fields computed by meshfree method with distance fields: very strong temperature-dependence of thermal conductivity, Dirichlet boundary conditions, and time step $t = 1$ sec.

chosen as B-splines of the fourth degree distributed over a uniform rectangular grid. To validate the accuracy and convergence properties of the meshfree method with distance fields we compared solutions computed by meshfree method with distance with ANSYS 12.1 solutions.

Solutions were computed by meshfree method with distance fields using algorithms based on Oseen linearization scheme and Newton-Kantorovich linearization scheme. The linearization schemes applied to linearize the nonlinear terms in the equations that defined the problem leads to an iterative procedure. The convergence of iterative solutions is achieved when the difference between two consecutive solutions becomes sufficiently small ε (Equation 3.9). The iterations are stopped as soon as $\varepsilon \leq 10^{-6}$. For steady-state



(a) 5 sec (b) 15 sec

(c) 25 sec (d) 100 sec

Figure 3.19: Quasi-steady temperature fields computed by ANSYS 12.1: very strong temperature-dependence of thermal conductivity, Dirichlet boundary conditions, and time step $t = 1$ sec.

and transient problems with convective boundary conditions, we conducted experiments using a material with weak temperature-dependence of thermal conductivity, and a material with very strong temperature-dependency of thermal conductivity.

3.8.1 Experiment 9: Steady-state with weak temperature dependency of thermal conductivity

The meshfree method with distance fields was applied to modeling heat transfer in the benchmark problem Figure 3.1(a) made from a material with weak temperature-dependency of thermal conductivity (alumina).

The convective boundary conditions are formulated by setting the temperature of the medium surrounding the outer boundary ω_1 to 320 °K, and setting the temperature

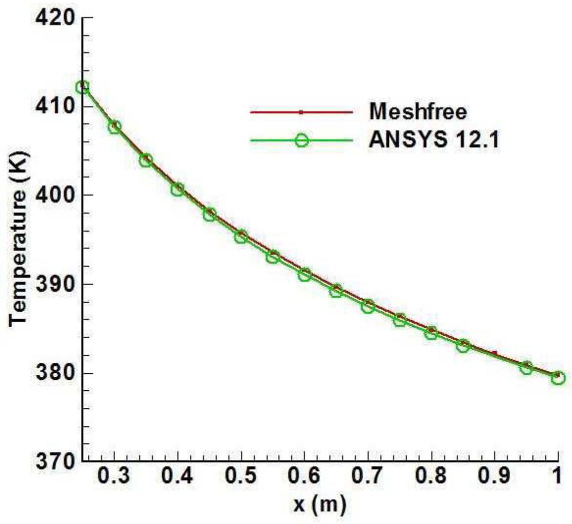


Figure 3.20: Temperature distribution in the benchmark problem Figure (3.1a) computed by meshfree method with distance fields and ANSYS 12.1: steady-state problem with convective boundary conditions and weak temperature-dependency of thermal conductivity.

of the medium surrounding the inner boundary ω_2 to 650 °K. Heat transfer on the boundaries is at 13.1 W/m^2-K. We observed the behavior of the meshfree method with distance fields using algorithms based on Oseen and Newton-Kantorovich linearization schemes. The plots shown in Figure 3.21 illustrate that both schemes converges very fast to the solution.

Table 3.5: Relative difference between meshfree method with distance fields and ANSYS 12.1 solutions: steady-state with weak temperature-dependence of thermal conductivity and convective boundary conditions.

Location (x, y)	Oseen Meshfree (K)	Newton-Kantorovich Meshfree (K)	ANSYS 12.1 (K)	Relative Difference Oseen-Ansys (%)	Relative Difference Kantorovich-Ansys (%)
0.4, 0	400.983807	400.987542	400.67	0.0783	0.0792
0.5, 0	395.776579	395.77294	395.33	0.1128	0.1119
0.6, 0	391.545055	391.546676	391.06	0.1239	0.1243
0.7, 0	387.96168	387.962464	387.5	0.1190	0.1192
0.8, 0	384.861649	384.861931	384.5	0.0940	0.0940

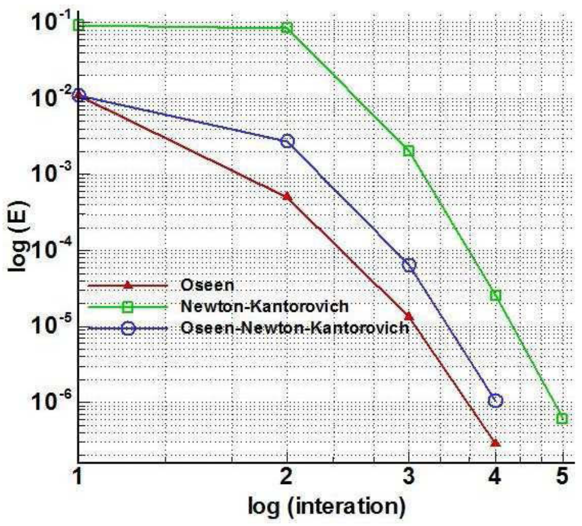


Figure 3.21: Convergence in terms of relative difference between two consecutive solutions ε (Equation 3.9). Convergence was observed for Oseen linearization, Newton-Kantorovich linearization, and both Oseen and Newton-Kantorovich applied in the linearization process. Steady-state with weak temperature-dependency and convective boundary conditions.

Accuracy of the computation by meshfree method with distance fields is confirmed by the experimental results shown in Table 3.5. The results illustrate that temperatures computed by meshfree method with distance fields and temperatures predicted by FEM ANSYS 12.1 exhibits almost no difference. Comparison of Figures 3.20(a) and (b) with Figure 3.20(c) demonstrate that the temperature distribution computed by meshfree method with distances fields is essentially identical to the temperature distributions produced by ANSYS 12.1.

3.8.2 Experiment 10: Steady-state with strong temperature dependency of thermal conductivity

The meshfree method with distance fields was applied to modeling heat transfer in the benchmark problem Figure 3.1(a) made from a material with very strong temperature-dependency of thermal conductivity (copper). The temperature-dependency of thermal conductivity data for cooper, used in this experiment, is taken from Figure 3.2(b). The convective boundary conditions are formulated by setting the temperature of the medium surrounding the outer boundary ω_1 to 1 °K, and setting the temperature of the medium surrounding the inner boundary ω_2 to 60 °K. Heat transfer on the boundaries is at 13.1 W/ m-K. We observed the behavior of the meshfree method with distance fields using algorithms based Oseen and Newton-Kantorovich linearization schemes.The plots shown in Figure 3.22 illustrate that both schemes converges very fast to the solution. Accuracy of the computation by meshfree method with distance fields is confirmed by the experimental results shown in Table 3.6.

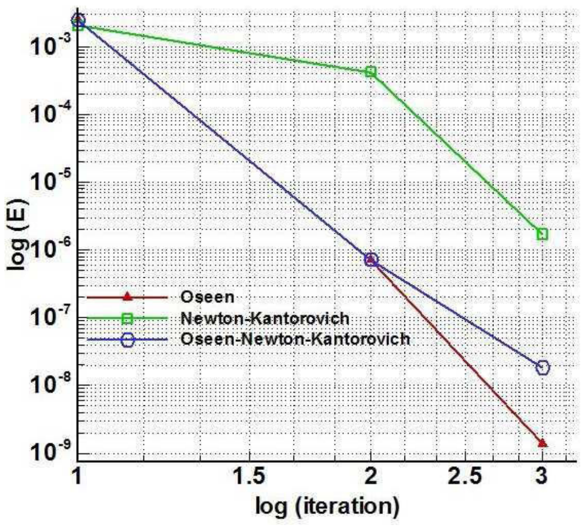


Figure 3.22: Convergence in terms of relative difference between two consecutive solutions ε (Equation 3.9). Convergence was observed for Oseen linearization, Newton-Kantorovich linearization, and both Oseen and Newton-Kantorovich applied in the linearization process. Steady-state with strong temperature-dependency and convective boundary conditions.

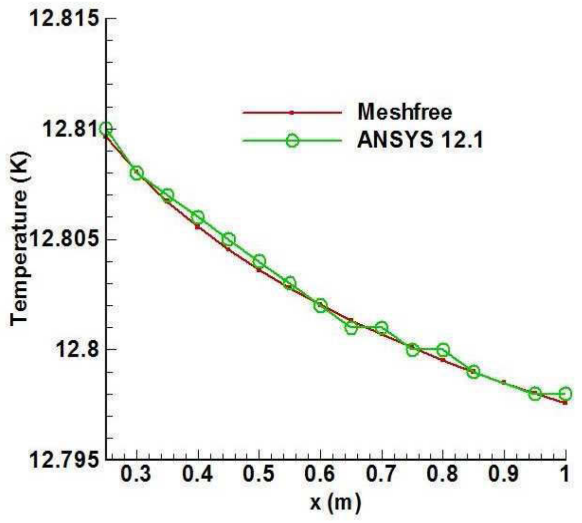


Figure 3.23: Temperature distribution in the benchmark problem Figure (3.1a) computed by meshfree method with distance fields and ANSYS 12.1: steady-state problem with convective boundary conditions and very strong temperature-dependency of thermal conductivity

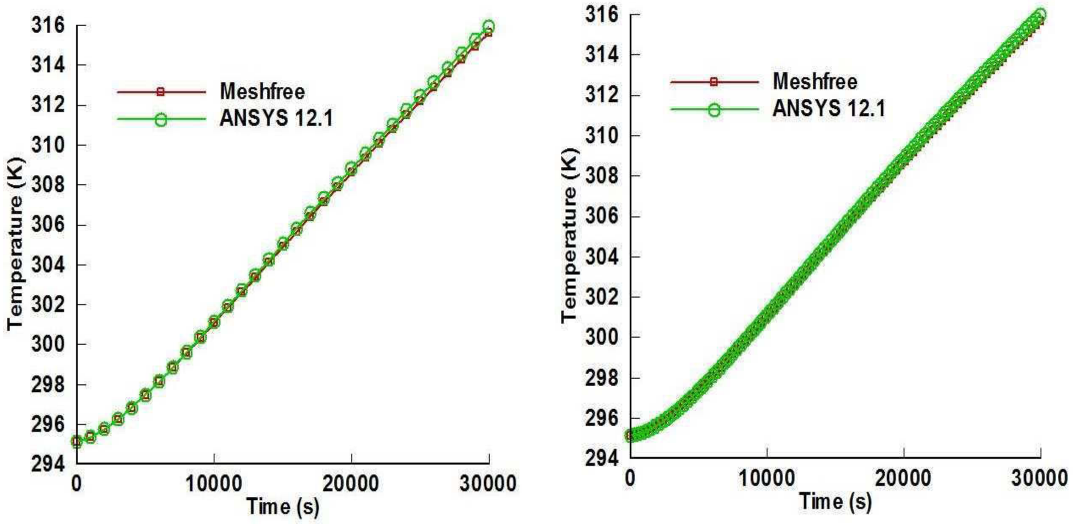
Table 3.6: Relative difference between meshfree method with distance fields and ANSYS 12.1 solutions.: Solutions obtained for steady-state with convective boundary conditions very strong temperature-dependence of thermal conductivity.

Location (x, y)	Oseen Meshfree (K)	Newton-Kantorovich Meshfree (K)	ANSYS 12.1 (K)	Relative Difference Oseen-Ansys (%)	Relative Difference Kantorovich-Ansys (%)
0.4, 0	12.805553	12.805553	12.801	0.0356	0.0356
0.5, 0	12.803614	12.803614	12.804	0.0030	0.0030
0.6, 0	12.80203	12.80203	12.802	0.0002	0.0002
0.7, 0	12.80069	12.80069	12.801	0.0024	0.0024
0.8, 0	12.79953	12.79953	12.8	0.0037	0.0037

The results illustrate that temperatures computed by meshfree method with distance fields and temperatures predicted by FEM ANSYS 12.1 exhibits almost no difference. Comparison of Figures 3.23(a) with Figure 3.23(b) demonstrate that the temperature distribution computed by meshfree method with distances fields is essentially identical to the temperature distributions produced by ANSYS 12.1.

3.8.3 Experiment 11: Transient with weak temperature dependency of thermal conductivity

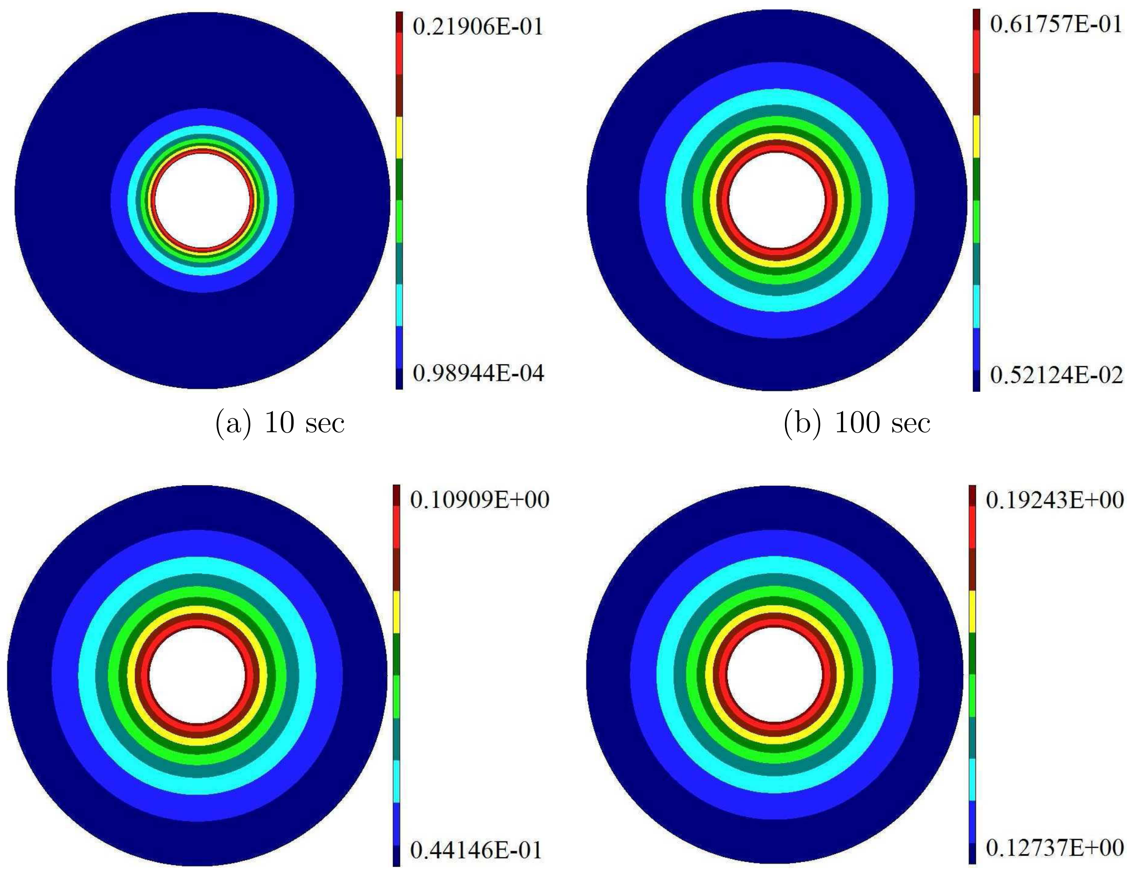
The convective boundary conditions are formulated by setting the temperature of the medium surrounding the outer boundary ω_1 to 320 °K, and setting the temperature of the medium surrounding the inner boundary ω_2 to 650 °K. Heat transfer on the boundaries is at 13.1 W/m^2-K.



(a) (b)

Figure 3.24: Comparing time evolution of the temperature at location $x = 0.8, y = 0.0$ predicted by meshfree method with distance fields and ANSYS 12.1: for two different time steps, convective boundary conditions and weak temperature-dependence of thermal conductivity.

Figure 3.24 shows temperature prediction in the benchmark problem with weak temperature-dependence of thermal conductivity, $\lambda(T)$ given in Figure 3.2, density $\rho = 3960$ kg/m^3 and specific heat $c = 850$ J/kg-K. The initial temperature is $T = 215.15$ °K. Figures 3.24(a) and (b) show the time evolution of the temperature at location $x = 0.8, y = 0.0$, computed by meshfree method with distance fields and by FEM software ANSYS 12.1 for two different time steps: 1000 seconds in Figure 3.24(a) and 300 seconds in Figure 3.24(b). Figure 3.24 illustrate that time evolution of the temperature computed by meshfree method with distance fields is essentially identical that of ANSYS 12.1.



(a) 10 sec (b) 100 sec

(d) 400 sec (c) 1000 sec

Figure 3.25: Quasi-steady temperature fields computed by meshfree method with distance fields: very strong temperature-dependence of thermal conductivity, convective boundary conditions, and time step $t = 10$ sec.

3.8.4 Experiment 12: Transient with strong temperature dependency of thermal conductivity

The convective boundary conditions are formulated by setting the temperature of the medium surrounding the outer boundary ω_1 to 1 °K, and setting the temperature of the medium surrounding the inner boundary ω_2 to 60 °K. Heat transfer on the boundaries is at 13.1 W/m^2-K. Figure 3.26 shows temperature prediction at location $x = 0.8, y = 0.0$ in the benchmark problem with very strong temperature-dependence of thermal conductivity, $\lambda(T)$ taken from data given in Figure 3.2(b), density $\rho = 8300$ kg/m^3 and specific heat $c = 385$ J/kg-K. The problem is defined to have an initial uniform temperature of $T = 0.0$ °K. Figures 3.26(a) and (b) show the time evolution of the

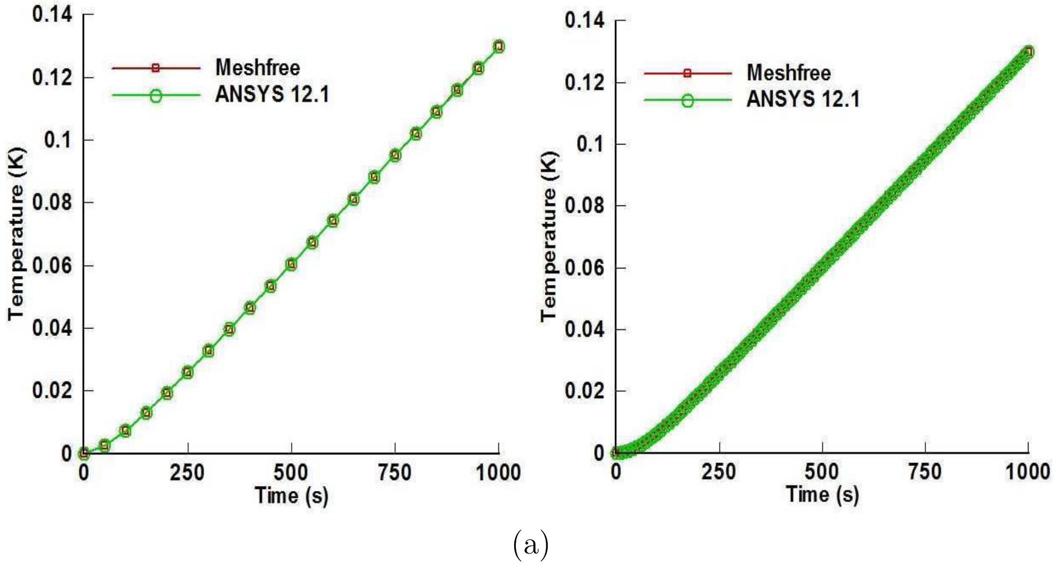
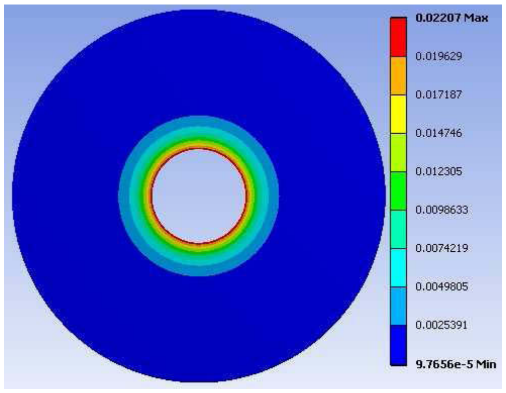


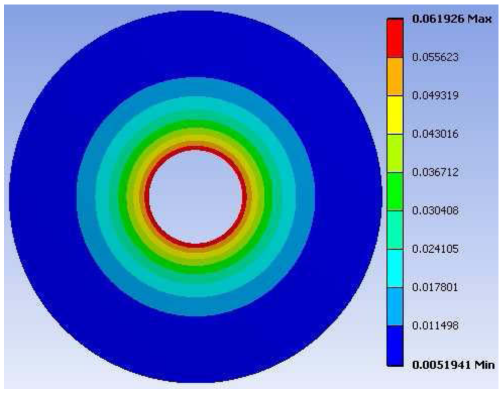
Figure 3.26: Comparing time evolution of the temperature at location $x = 0.8, y = 0.0$ predicted by meshfree method with distance fields and ANSYS 12.1: for two different time steps, convective boundary conditions and material with very strong temperature-dependence of thermal conductivity.

temperature at location $x = 0.8, y = 0.0$, computed by meshfree method with distance fields and by FEM software ANSYS 12.1 for two different time steps: 50 seconds in Figure 3.26(a) and 10 seconds in Figure 3.26(b).

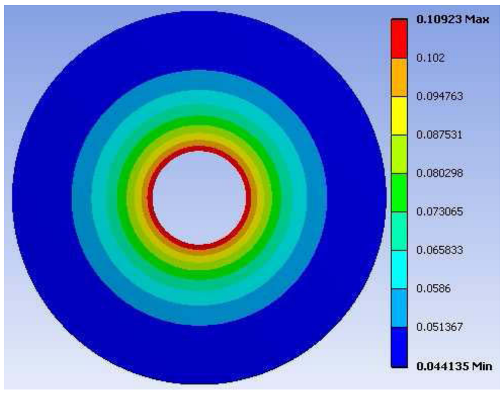
The plots in Figure 3.26 illustrate that time evolution of the temperature computed by meshfree method with distance fields is essentially identical that of ANSYS 12.1. Comparison of Figure 3.25 with Figure 3.27 shows almost no difference between the quasi-steady temperature fields computed by meshfree method with distances and the quasi-steady temperature fields predicted by ANSYS 12.1. This experiment also confirm the accuracy of the meshfree method with distance fields computation to obtain solution of nonlinear transient heat transfer problem using material with strong temperature-dependency of thermal conductivity and convective boundary conditions.



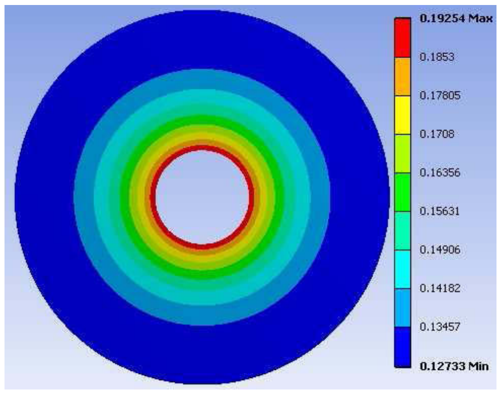
(a) 10 sec



(b) 100 sec



(d) 400 sec



(e) 1000 sec

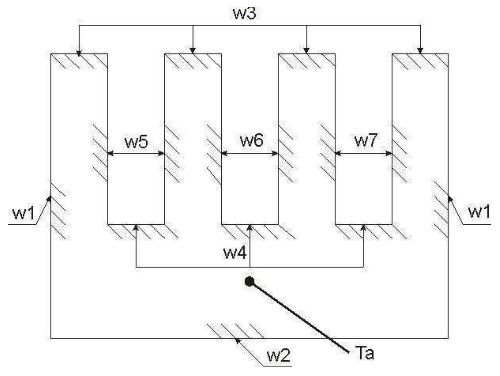
Figure 3.27: Quasi-steady temperature fields computed by ANSYS 12.1: very strong temperature-dependence of thermal conductivity, convective boundary conditions, and time step $t = 10$ sec.

CHAPTER 4

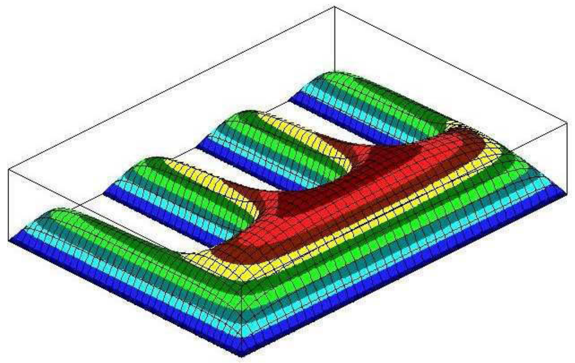
DISCUSSION AND CONCLUSION

In this thesis work, we successfully developed and implemented numerical algorithms for solving nonlinear transient heat transfer problems, using meshfree method with distance fields. The results of the numerical experiments presented in chapter 3 illustrate the accuracy and convergence of the numerical algorithms. We validated the numerical computations on a simple two-dimensional benchmark problem Figure 3.1(a) since analytical solution of this benchmark problem is easily available. However, the meshfree method with distance fields algorithms as implemented can be applied to any two-dimensional geometry. Solutions obtained by meshfree method with distance fields and commercial FEM software ANSYS 12.1, were observed to exhibits almost no difference.

Meshfree method with distance field requires that solution of the nonlinear transient heat transfer problem equation must incorporate the analytic information about the boundary conditions, as well as geometric information about the boundaries where these conditions are specified. This feature of meshfree method with distance fields allows for boundary conditions to be satisfied exactly. The equations to be solved contains nonlinear terms which must be linearized. Since the way of linearization can significantly affect the of convergence towards the final solution, choice of an appropriate linearization method was therefore important. We developed numerical algorithms with two different linearization schemes applied to linearize the nonlinear terms in the equations; Oseen linearization scheme and Newton-Kantorovich linearization scheme, which both lead to an iterative procedure. The iterative procedure involves updating the nonlinear terms with values evaluated using solutions obtained at the previous iteration. We assumed a constant value for the nonlinear terms at the initial iteration.



(a)



(b)

Figure 4.1: (a) Two dimensional representation of the geometric domain of the fin type heat exchanger; (d) the corresponding approximate distance field.

As illustrated in Figure 3.14 we observed that Newton-Kantorovich scheme enjoys rapid convergence when applied in problem with Dirichlet boundary conditions and the temperature-dependence of thermal conductivity is weak. However, for a problem with Dirichlet boundary conditions and the temperature-dependence of thermal conductivity is very steep, the solution diverged when Newton-Kantorovich scheme is applied. The solution was very slow to converged with Oseen scheme applied. We know that because a constant value is assumed for the thermal conductivity at the initial iteration, the computed solution at this step may be far from the true solution.

It is well known that Newton method can potentially diverge from the solution if the initial solution in the iterative process is sufficiently far from the desired solution. Since Oseen scheme was slow to converged and Newton-Kantorivich scheme resulted in divergence, we implemented algorithm that incorporated both schemes into the linearization process. This algorithm applies Oseen linearization scheme at the initial iteration. Newton-Kantorovich scheme is then applied when the difference between two consecutive solutions is computed to be less than one $\varepsilon < 1$. We observed that by applying both linearization schemes into the linearization process, we developed algorithm that produced fast converged solutions and insured convergence.

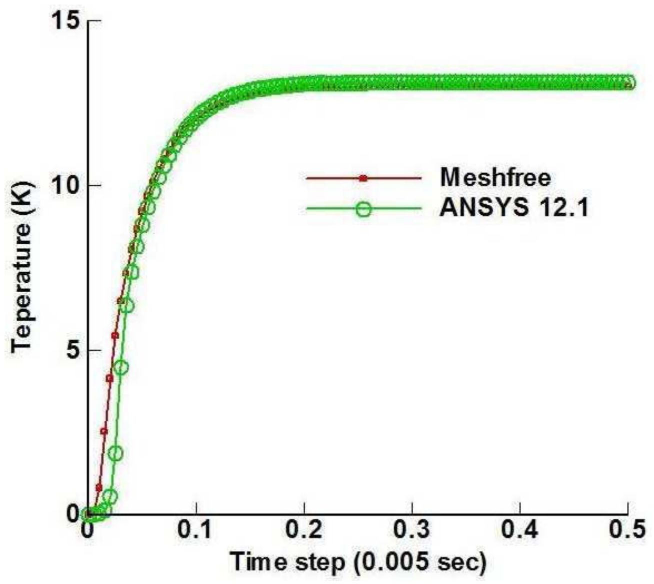


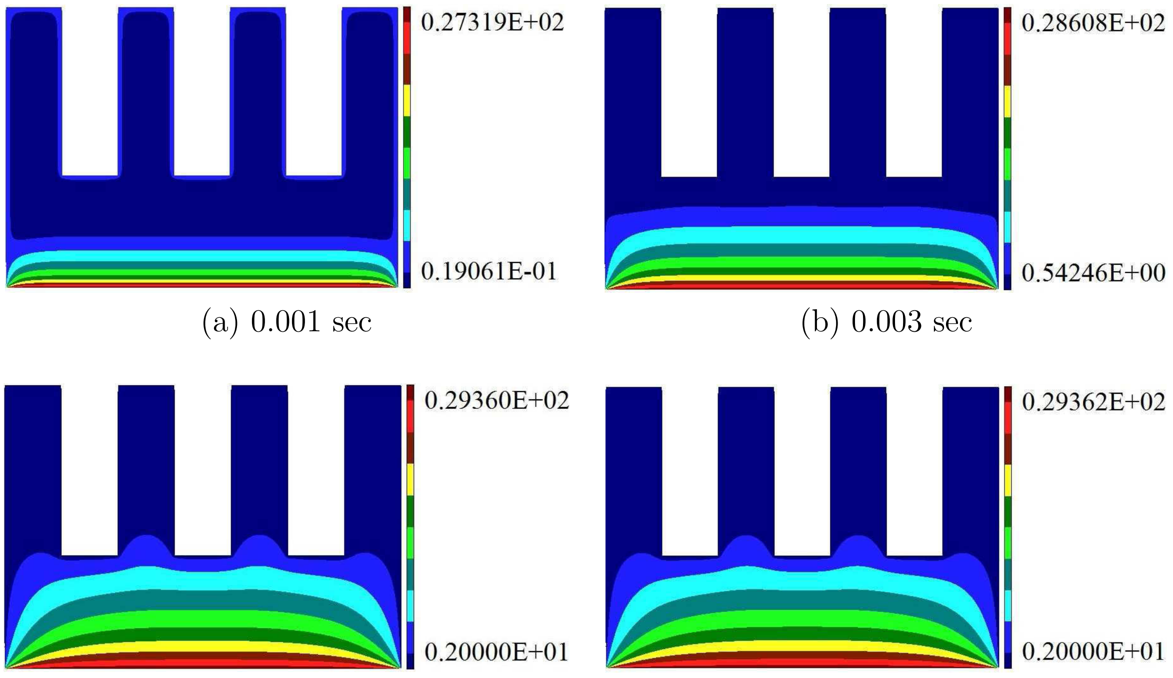
Figure 4.2: Comparing time evolutions of temperatures predicted by meshfree method with distance fields and ANSYS 12.1.

To demonstrate that the meshfree method as implemented can be applied to geometric domain other than the geometric domain of the benchmark problem, we applied the meshfree method with distance fields to modeling nonlinear transient heat transfer in a fin type heat exchanger. Figure 4.1(a) shows the geometric primitives that defined the geometric domain of the fin type heat exchanger. These primitives, as we described in chapter 3, can be combined using R--functions set operations $\wedge_0$ and $\vee_0$ to define the two-dimensional domain Ω as:

$$\omega = (\omega 1 \wedge_0 \omega 2 \wedge_0 \omega 3) \wedge_0 (\omega 4 \vee_0 \omega 5) \wedge_0 (\omega 4 \vee_0 \omega 6) \wedge_0 (\omega 4 \vee_0 \omega 7),$$

where the symbols $\wedge_0$ and $\vee_0$ denote R_0-conjunction and R_0-disjunction respectively. The domain representation can be translated into the approximate distance field ω shown in Figure 4.1(b) using R--functions.

The problem is formulated as a nonlinear transient heat transfer problem with Dirichlet boundary conditions. The heat exchanger is made from a material with very steep temperature-dependence of thermal conductivity. Boundary conditions are formulated by a prescribed temperature of 65 °K on the lower boundary $\omega 2$, and a temperature



(a) 0.001 sec (b) 0.003 sec

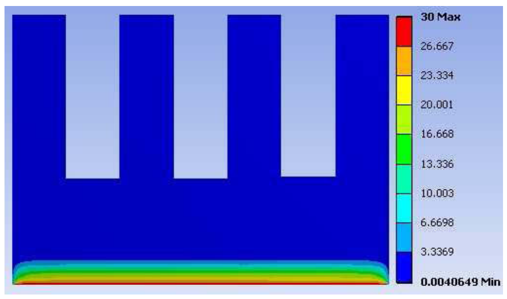
(d) 0.25 sec (e) 0.5 sec

Figure 4.3: Quasi-steady temperature fields computed by meshfree method with distance fields.

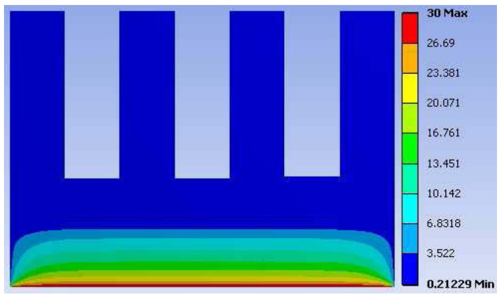
of 2 °K prescribed on all other boundaries. Figure 4.2 shows time evolution of the temperature at point denoted by Ta in Figure 4.1 of the heat exchanger. The material in the region has very steep temperature-dependence of thermal conductivity $\lambda(T)$ as described by the plot shown in Figure 3.2(b), density $\rho = 8300$ kg/m^3 and specific heat c = 850 J/kg-K. The problem is defined to have an initial uniform temperature of $T = 0.0$ K.

Time evolution of the temperature plots shown in Figure 4.2 illustrate good agreement between the temperatures computed by meshfree method with distance fields and ANSYS 12.1. Comparison of Figure 3.25 with Figure 3.27 shows almost no difference between the quasi-steady temperature fields computed by meshfree method with distances and the quasi-steady temperature fields predicted by ANSYS 12.1.

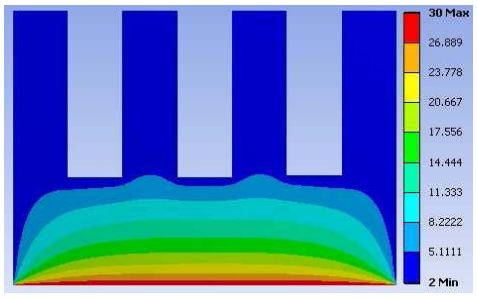
The Meshfree method with distance fields explicitly incorporates the temperature-dependent materials properties into the approximate solution and it enables the all



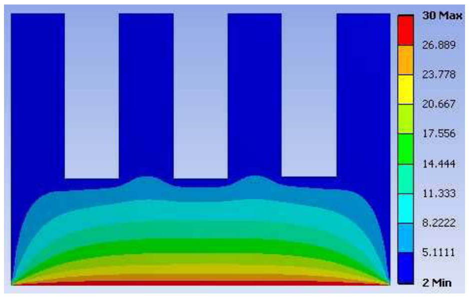
(a) 0.001 sec



(b) 0.003 sec



(d) 0.25 sec



(e) 0.5 sec

Figure 4.4: Quasi-steady temperature fields computed by ANSYS 12.1.

prescribed boundary conditions to be satisfied exactly. Although we modeled only two-dimensional problems in this thesis work, the algorithms that we developed in this thesis work can be easily extended solve three-dimensional problems. Since meshfree method with distances fields does not require meshing that have to conform to the geometric domain, we can potentially of save a lot of resource time by using our developed algorithms to obtain solution of nonlinear transient heat transfer problems.

REFERENCES

[1] A. Biswas and V. Shapiro. Approximate distance fields with non-vanishing gradients. *Graphical Models*, 66:133–159, 2004.

[2] G. Comini, S Del Guidance, R. W. Lewis, and O. C. Zienkiewicz. Finite element solution on nolinear heat conduction problems with special reference to phase change. *International Journal for Numerical Methods in Engineering*, 8:613–624, 1974.

[3] C. A. Duarte and J. T. Oden. An h-p adaptive methods using clouds. *Computer Methods in Applied Mechanics and Engineering*, 139:237–262, 1996.

[4] M. Freytag, V. Shapiro, and I. Tsukanov. Field modeling with sampled distance. *Computer-Aided Design*, 38:87–100, 2006.

[5] C. Y. Ho, R. W. Powell, and P. E. Liley. Thermal conductivity of elements. *Journal of Physical and Chemical Reference Data*, 3(1), 1974.

[6] L. V. Kantorovich and V. I. Krylov. *Approximate Methods of Higher Analysis.* Interscience Publishers, 1958.

[7] Y. Krongauz and T. Belytschko. Enforcement of essential boundary conditions in meshless approximations using finite elements. *Computer Methods in Applied Mechanics and Engineering*, 131:133–145, 1996.

[8] P. Krysl. *A Pragmatic Introduction to Finite Element Method for Thermal and Stress Analysis.* World Scientific Publishing Co. Pte. Ltd, 2008.

[9] W. K. Liu, S. Jun, S. Li, J. Adee, and T. Belytschko. Reproducing kernel particle methods for structural mechanics. *International Journal for Numerical Methods in Engineering.*, 38:1655–1679, 1995.

[10] Y.Y. Lu, T. Belytschko, and L. Gu. A new implementation of the element-free Galerkin. *Computer Methods in Applied Mechanics and Engineering*, 113:397–414, 1994.

[11] L. Lucy. A numerical approach to testing the fission hypothesis. *Astron. J.*, 82:1013–1024, 1977.

[12] T. J. Martin and G. S. Dulikravich. Inverse determination of temperature-dependent thermal conductivity using steady surface data on orbitrary objects. *Journal of Heat Transfer*, 122:450–459, 2000.

[13] J. M. Melenk and I. Babuska. The partition of unity finite element method: Basic theory and applications. *Computer Methods in Applied Mechanics and Engineering*, 139:289–314, 1996.

[14] B. Nayroles, G. Touzot, and P. Villon. Generalizing the finite element method: diffuse approximation and diffuse elements. *Computational Mechanics*, 10:307–318, 1992.

[15] M. N. Ozisic. *Heat Conduction.* John Wiley and Sons, 1993.

[16] V. L. Rvachev and T. I. Sheiko. R-functions in boundary value problems in mechanics. *Applied Mechanics Reviews*, 48(4):151–188, 1995.

[17] V. L. Rvachev, T. I. Sheiko, V. Shapiro, and I. Tsukanov. On completeness of RFM solution structures. *Computational Mechanics*, 1999. Accepted for publication in the special issue on meshfree methods.

[18] G. E. Schneider. A physical aaproach to the finite-difference solution of the conduction equation in generalized coordinates. *Numerical Heat Transfer, Part A: An iternational journal of computation and methodology.*, 5(1):1–19, 1982.

[19] J.A. Sethian. *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Material Sciences.* Cambridge University Press, 1996.

[20] V. Shapiro. Theory of R-functions and applications: A primer. Report TR91-1219, Computer Science Department, Cornell University, 1991.

[21] V. Shapiro. Real functions for representation of rigid solids. *Computer-Aided Goemetric Design*, 11(2):153–175, 1994.

[22] V. Shapiro. Semi-analytic geometry with r-functions. *Acta Numerica*, 12:239–303, 2007.

[23] V. Shapiro and I. Tsukanov. Implicit functions with guaranteed differential properties. In *Fifth ACM Symposium on Solid Modeling and Applications*, Ann Arbor, MI, 1999.

[24] V. Shapiro and I. Tsukanov. Meshfree simulation of deforming domains. *Computer-Aided Design*, 31(7):459–471, 1999.

[25] T. H. Sheu and R. K. Lin. Newton linearization of incompressible navier-stoke equations. *International Journal for Numerical Methods in Engineering*, 44(3):297–312, 2004.

[26] L. C. Wrobel and M. H. Aliabadi. *Boundary Element Method.* John Wiley and Sons, 2002.

APPENDIX

Appendix 1: Analytical Solution of the Benchmark Problem: Steady-State Problem with Dirichlet Boundary Conditions

```
#include "rfms.h"

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <math.h>

int pre(void);

tuple omega(void);

double om(void);

double fa(int flag, int i, int j);

double fb(int flag, int i);

double fu(void);

void  u0(void);

tuple u1(void);

tuple *w1, *w2;

tuple_seq *p;

// Compute exact solution
```

```
double u_exact(void)

{

        double x, y;

        double c1, c2, r1, r2, d1, d2;

        r1 = 0.25;  r2 = 1;  d1 = 40;  d2 = 1;

        c2=(((d1*log(r2)))-(d2*log(r1)))/((log(r2))-(log(r1)));

        c1=((d1-(c2))/log(r1));

        x = GetArgumentX(1);    y = GetArgumentX(2);

        return c1 * log( sqrt(x*x + y*y) ) + c2;

}

// Error computation

double u_diff2(void)

{

        double temp;

        temp = (u_exact() - fu());

        return temp * temp;

}


double u_exact2(void)
```

```
{
        double temp = u_exact();
        return temp * temp;
}
int main(void)
{
        timerclass T;
        T.ShowTimeBegin();
// Allocation of the data structure for automatic differentiation
        SetTupleDimensionOrder(2,1); //2D, max order = 1;
// Defining the bounding box
        double xmin, ymin, xmax, ymax;
        double xmin_ext, ymin_ext, xmax_ext, ymax_ext;
        xmin  =  ymin  =  -1.0;
        xmax  =  ymax  =  1.0;
        SetBoundingBox2D( xmin, ymin, xmax, ymax );
        w1 = new tuple;
        w2 = new tuple;
//      Defining information about B-splines: degree, grid
```

```
          int ns, nx, ny  gauss,  n_random;

          ns = 4;   // degree of B-splines   nx = ny = 22;  // grid nx x ny

          n_gauss = 5;    n_random= 5;

//        Allocation of the matrix and vector

          matrixclass a;

          vectorclass b;

          a.CreateMatrix(M_BANDED_SYMMETRIC, ns, nx, ny);

          b.CreateVector(ns, nx, ny);

//  Defining information about basis functions

          tuple_seq P(1); p = &P; //An empty entry; (1) means the number of sequences

//  The first sequence is B-splines of degree ns, with grid nx*ny

          P.SetBsplines2DDegree(0, ns, nx, ny);     // 0's sequence

//  Define which matrix and vector will be assembled by the integration procedures

          SetMatrixPointer( &a );    SetVectorPointer( &b );

          AdjustBoundingBox2D(xmin, ymin, xmax, ymax,   xmin_ext, ymin_ext,

          xmax_ext,   ymax_ext);

          quadtreeclass tree(nx, ny, xmin_ext, ymin_ext, xmax_ext, ymax_ext);

          tree.BuildTree(n_gauss,0.5, n_random, pre);

//        Assemble matrix and vector using functions fa and fb
```

```
        tree.IntegrateMV(fa,fb);

        a.Solve( b ); // on return b contains values of C_i

//  Compute  L2-norm of the error

        double I1, I2, error;

        tree.Integrate(u_diff2);

        I1 = sqrt( GetIntegralValue() );

        tree.Integrate(u_exact2);

        I2 = sqrt( GetIntegralValue() );

        error = I1/I2 * 100;

        //  save results

        filexy("u_approx",199, 199, xmin,ymin, xmax, ymax, fu, om);

        T.ShowTime();

        return(0);

}

//Point membership classification function

int pre(void)

{

        if( om() >= 0.0)

        {
                return 1;
        }
        else
        {
                return -1;
        }
}
double om(void)
{
        return omega().GetValue();
}
// The omega function
tuple omega(void)
{
        *w1 = circle( 0,0,1 );   *w2 = mcircle(0,0,0.25);
         return ( *w1 ) & (*w2);
}
//Solution structure
void u0(void)
```

```
{
        p->Compute(0);
        p->Mult(omega());
        return;
}
tuple u1(void)
{
        tuple f1, f2;   f1 = 1;  f2 = 40;
        omega();
        return paste((*w1),(*w2), f1,f2);
}
// Matrix function
double fa(int flag, int i, int j)
{
        double result1,result2;
        if( flag == 0 ) // new integration point
        {
                u0();
        }
//  Computation of the dot product of the gradients

        result1 =   dx_direct( p->GetTuple(0,i), 1)* dx_direct( p->GetTuple(0,j), 1);

        result2 =   dx_direct( p->GetTuple(0,i), 2)*dx_direct( p->GetTuple(0,j), 2);

        return (result1 + result2);

}

// Vector function

double fb(int flag, int i)

{

        double result1,result2;

        static tuple uu1(GetTupleMaxOrder());

        if( flag == 0 )

        {

                u0();

                uu1 = u1();

        }

        result1 =  dx_direct( p->GetTuple(0,i), 1) *dx_direct( &uu1, 1 );

        result2 =  dx_direct( p->GetTuple(0,i), 2 )* dx_direct( &uu1, 2 );

        return -(result1 + result2);

}
```

```
// SOLUTION

double fu(void)

{

        static vectorclass *c;

        c = GetVectorPointer();

        u0();

return (sum( c, u1(), *p ).GetValue());

}
```

Appendix 2: Solution of Nonlinear Transient Heat Conduction Problem with Dirichlet Boundary Conditions

```
#include "rfms.h"

#include <stdio.h>

#include <math.h>

int pre(void);

tuple omega(void);

double fa(int flag, int i, int j);

double fb(int flag, int i);

double fu(void);

double om(void);
```

```
void u0(void);

tuple u1(void);

double fa_Initial(int flag, int i, int j);

double fb_Initial(int flag, int i);

double fa_InitialT(int flag, int i, int j);

double fb_InitialT(int flag, int i);

double dirLambda_prev(void);

double fu_initial(void);

double lambda_prev(void);

tuple *w1, *w2;

tuple_seq *p;

vectorclass *c_previous, *c_current,*c_previousT, *c_currentT;

double t, dt, lambda, Rho, Cp, Cp_Rho, TempMax, TempMin;;

SplineInterpolation *lambda_table;



double regularization_parameter = 0.0; // set zero to start computation Oseen

int main(void)

{

        timerclass T;
        T.ShowTimeBegin();

// Allocation of the data structure for automatic differentiation

        SetTupleDimensionOrder(2,1); //2D, max order = 1;

// Defining the bounding box

        double xmin, ymin, xmax, ymax;

        double xmin_ext, ymin_ext, xmax_ext, ymax_ext;

        xmin = ymin = -1.0;    xmax = ymax = 1.0;

        SetBoundingBox2D( xmin, ymin, xmax, ymax );

        w1 = new tuple;  w2 = new tuple;

//  Thermal conductivity (W/m-K) data for Alumina (Temp in K)

        double Temp[10], K[10];

        int number_of_data_pts = 5;

        Temp[0] = 298.15;     K[0] = 46.0;

        Temp[1] = 400.15;     K[1] = 32.3;

        Temp[2] = 500.15;     K[2] = 24.2;

        Temp[3] = 600.15;     K[3] = 18.9;

        Temp[4] = 800.15;     K[4] = 13.0;

        Rho = 3960;  Cp  = 850;

lambda_table = new SplineInterpolation(Temp, K, number_of_data_pts);
```

```
          TempMin = Temp[0]; TempMax = Temp[32]; lambda = K[0];

          dt = 2;  Cp_Rho = 1/(Cp*Rho);

//        Defining information about B-splines: degree, grid

          int ns, nx, ny, ngauss, random;

          ns = 4;        // degree of B-splines

          nx = ny = 61; // grid nx x ny

          ngauss = 5; random = 5;

//        Allocation of the matrix and vector

          matrixclass a;

          vectorclass b, C_previous, C_current, C_previousT, C_currentT;



          a.CreateMatrix(M_BANDED, ns, nx, ny);

          b.CreateVector(ns, nx, ny);

//  Assign C for time iteration

          C_previousT.CreateVector(ns, nx, ny); C_currentT.CreateVector(ns, nx, ny);

          c_previousT = &C_previousT; c_currentT = &C_currentT;

//        Assign C for Conductivity iteration

          C_previous.CreateVector(ns, nx, ny); C_current.CreateVector(ns, nx, ny);

          c_previous = &C_previous; c_current = &C_current;
```

```
//  Defining information about basis functions

          tuple_seq P(1); p = &P; //An empty entry; (1) means the number of sequences

//  The first sequence is B-splines of degree ns, with grid nx*ny

          P.SetBsplines2DDegree(0, ns, nx, ny); // 0's sequence



//  Define which matrix and vector will be assembled by the integration procedures

          SetMatrixPointer( &a );

          SetVectorPointer( &b );

          AdjustBoundingBox2D(xmin, ymin, xmax, ymax,   xmin_ext, ymin_ext,

          xmax_ext, ymax_ext);

          quadtreeclass tree(nx, ny, xmin_ext, ymin_ext, xmax_ext, ymax_ext);

// Create quad-tree decomposition of the space using defined gaussian integration points;

          tree.BuildTree(random, 0.5, ngauss, pre);

//        Assemble initial matrix and vector for first "time" iteration

          tree.IntegrateMV(fa, fb_Initial);

          a.Solve(b); // on return b contains values of C_i

          C_currentT = b;

//        Nonlinear loop for the first iteration

          double err;  int k;
```

```
C_previous = C_currentT;

        do{

                        a.SetZero();

                        b.SetZero();

                        tree.IntegrateMV(fa, fb_Initial);

                //      Solve linear algebra problem

                        a.Solve(b);

                        C_current = b;

err = C_current.RelativeErrorEstimation( C_previous, V_NORM_2 );

                        C_previous = C_current;

// Apply Oseen or Newton-Kantorovich Linearization

                        if (err < 1.0)

                        {

                                regularization_parameter = 1.0;

                        }

                        else

                        {

                                regularization_parameter = 0.0;

                        }
                } while( err > 0.00001 ); // Setting convegence criteria

            filexy( 299,299, xmin,  ymin,  xmax,  ymax, fu, om);

//      Stepping through time

        for(k=1;k<50;k++)

        {

            C_previousT = C_current;

                do{

                    a.SetZero();

                    b.SetZero();

                    tree.IntegrateMV(fa, fb);

                    a.Solve(b);

                    C_current = b;

        err = C_current.RelativeErrorEstimation( C_previous, V_NORM_2 );

           C_previous = C_current;

            if (err < 1.0)

                    {

                        regularization_parameter = 1.0;

                    }

                    else
```

```
                        {
                                regularization_parameter = 0.0;
                        }
                } while( err > 0.00001 ); // Setting convegence criteria
                filexy( 299,299, xmin,  ymin,  xmax,  ymax, fu, om);
        }
    T.ShowTime();
    return(0);
}
//Point membership classification function
int pre(void)
{
    if( om() >= 0.0)
    {
        return 1;
    }
    else
    {
        return -1;
        }

}


double om(void)

{

        return omega().GetValue();

}

// The omega function

tuple omega(void)

{

        *w1 = circle( 0,0,1 ); *w2 = mcircle(0,0,0.25);

        return ( *w1 ) & (*w2);

}

// Initial temperature distribution

double fu_Initial(void)

{

        double x, y;

        x = GetArgumentX(1);

        y = GetArgumentX(2);
        return 0.0;

}

//Solution structure

void u0(void)

{

        p->Compute(0);

        p->Mult(omega());

        return;

}

tuple u1(void)

{

        tuple f1, f2; f1 = 1; f2 = 60;

        omega();

        return paste((*w1),(*w2), f1,f2);

}

// Matrix function for initial iteration

double fa_InitialT(int flag, int i, int j)

{

        double result1,result2, result3;
```

```
	if( flag == 0 ) // Compute basis function at the point new integration point

	{

		u0();

	}

//  Computation of the dot product of the gradients

result1 =   dx_direct( p->GetTuple(0,i), 1)*dx_direct( p->GetTuple(0,j), 1);

result2 =   dx_direct( p->GetTuple(0,i), 2)*dx_direct( p->GetTuple(0,j), 2);

result3 =   dx_direct( p->GetTuple(0,i), 0)*dx_direct( p->GetTuple(0,j), 0);

return Cp_Rho*dt*(result1 + result2)*lambda + result3;

}

double fb_InitialT(int flag, int i)

{

	double result1,result2;

	static tuple uu1(GetTupleMaxOrder());

	static double prev;

	if( flag == 0 )

	{

		u0();

		uu1 = u1();
```

```
                prev = fu_Initial() - dx_direct(&uu1, 0);

        }

        result1 =  dx_direct( p->GetTuple(0,i), 1) * dx_direct( &uu1, 1 );

        result2 =  dx_direct( p->GetTuple(0,i), 2 )*  dx_direct( &uu1, 2 );

        return -Cp_Rho*dt*(result1 + result2)*lambda + dx_direct( p->GetTuple(0,i), 0 )

        * prev;

}

double fa(int flag, int i, int j)

{

        double result1,result2, result3, result4, result5;

        static double lambda_Tprev;

        static double dirLambda_Tprev; // Change in lambda with repect to temp.

        static tuple fu_prev(GetTupleMaxOrder());

        if( flag == 0 ) // new integration point

        {

                u0();

                lambda_Tprev = lambda_prev();

                dirLambda_Tprev = dirLambda_prev();

                // compute solution at previous iteration
                fu_prev = sum(c_previous, u1(), *p );

        }

// Computation of the dot product of the gradients

result1 =   dx_direct( p->GetTuple(0,i), 1)* dx_direct( p->GetTuple(0,j), 1);

result2 =   dx_direct( p->GetTuple(0,i), 2)*dx_direct( p->GetTuple(0,j), 2);

result3 =   dx_direct( p->GetTuple(0,i), 0)*dx_direct( p->GetTuple(0,j), 0);

result4 =   dx_direct( p->GetTuple(0,i), 1)*dx_direct(&fu_prev, 1);

result5 =   dx_direct( p->GetTuple(0,i), 2)*dx_direct( &fu_prev, 2);

return  Cp_Rho*dt*(result1 + result2)*lambda_Tprev + result3 +

                  (

                Cp_Rho*dt*(result4 + result5)*dirLambda_Tprev * dx_direct( p-
                >GetTuple(0,j), 0)

                  ) * regularization_parameter;

}

double fb_Initial(int flag, int i)

{

        double result1,result2, result3, result4;

        static tuple uu1(GetTupleMaxOrder());

        static tuple fu_prev(GetTupleMaxOrder());
        static double prev, lambda_Tprev, dirLambda_Tprev;

        if( flag == 0 )

        {

                u0();

                uu1 = u1();

                lambda_Tprev = lambda_prev();

                prev = fu_Initial() - dx_direct(&uu1, 0);

                dirLambda_Tprev = dirLambda_prev();

                fu_prev = sum(c_previous, u1(), *p );

        }

        result1 =  dx_direct( p->GetTuple(0,i), 1) * dx_direct( &uu1, 1 );

        result2 =  dx_direct( p->GetTuple(0,i), 2 )* dx_direct( &uu1, 2 );

        result3 =  dx_direct( p->GetTuple(0,i), 1 )* dx_direct( &fu_prev, 1);

        result4 =  dx_direct( p->GetTuple(0,i), 2 )* dx_direct( &fu_prev, 2);

        return -Cp_Rho*dt*(result1 + result2)*lambda_Tprev

                + dx_direct( p->GetTuple(0,i), 0 ) * prev +

                  (
                - Cp_Rho*dt*(result3 + result4)*dirLambda_Tprev * dx_direct( &uu1, 0)
                + Cp_Rho*dt*(result3 + result4)*dirLambda_Tprev * dx_direct(
                &fu_prev, 0)
                  ) * regularization_parameter;
}
double fb(int flag, int i)
{
        double result1,result2, result3, result4;
        static double lambda_Tprev, dirLambda_Tprev, prev;;
        static tuple fu_prev(GetTupleMaxOrder());
        static tuple uu1(GetTupleMaxOrder());
        if( flag == 0 )
        {
                t = t - dt; // go to previous time
                double prev_u1 = u1().GetValue();
                t = t + dt;
                uu1 = u1();
                prev = prev_u1 - uu1.GetValue();
                u0();
                lambda_Tprev = lambda_prev();

                prev = (sum( c_previousT, *p ).GetValue()) + prev;

               dirLambda_Tprev = dirLambda_prev();

                fu_prev = sum(c_previous, u1(), *p );

        }

        result1 =  dx_direct( p->GetTuple(0,i), 1) *dx_direct( &uu1, 1 );

        result2 =  dx_direct( p->GetTuple(0,i), 2 )* dx_direct( &uu1, 2 );

        result3 =  dx_direct( p->GetTuple(0,i), 1 )* dx_direct( &fu_prev, 1);

        result4 =  dx_direct( p->GetTuple(0,i), 2 )* dx_direct( &fu_prev, 2);

        return  -Cp_Rho*dt*(result1 + result2)*lambda_Tprev +

                dx_direct( p->GetTuple(0,i), 0 ) * prev +

                  (

                   - Cp_Rho*dt*(result3 + result4)*dirLambda_Tprev *     dx_direct(
                   &uu1, 0)

                   + Cp_Rho*dt*(result3 + result4)*dirLambda_Tprev * dx_direct(
                   &fu_prev, 0)

                  ) * regularization_parameter;

 }

// Compute previous thermal conductivity k_prev
```

```
double lambda_prev(void)

{

        double fu_prev;

//  Compute basis functions at the point

        u0();

//  Compute the sum

        fu_prev= (sum(c_previous, u1(), *p ).GetValue());

        if(fu_prev < TempMin ) fu_prev = TempMin;

        if(fu_prev > TempMax ) fu_prev = TempMax;

        return lambda_table->Compute(fu_prev);

}

// Compute dirivative of lambda with respect to temperature obtained at previous iteration

double dirLambda_prev(void)

{

        double fu_prev;

        u0();

//  Compute the sum

        fu_prev= (sum(c_previous, u1(), *p ).GetValue());

        if(fu_prev < TempMin ) fu_prev = TempMin;
        if(fu_prev > TempMax ) fu_prev = TempMax;

        return lambda_table->ComputeDx1(fu_prev);

}


double fu(void)

{

//  Compute basis functions at the point

        u0();

//  Compute the sum

        return (sum( c_current, u1(), *p ).GetValue());

}
```

Appendix 3: Solution of Nonlinear Transient Heat Conduction Problem with Convective Boundary Conditions

```
#include "rfms.h"

#include <stdio.h>

#include <math.h>

int pre (void);

tuple omega(void);

tuple fi(void);
```

```
double om(void);

double fa(int flag, int i, int j);

double fa1(int flag, int i, int j);

double fb(int flag, int i);

double fb1(int flag, int i);

double fu(void);

double fb_initial(int flag, int i);

double fu_initial(void);

double fa_InitialT(int flag, int i, int j);

double fa1_initial(int flag, int i, int j);

double fb_initialT(int flag, int i);

double fb1_initial(int flag, int i);

double fu1(void);

double lambda_interpolate(void);

void u0(void);

tuple u1(void);

vectorclass *c;

vectorclass *c_previous, *c_current,*c_previousT, *c_currentT;

tuple_seq  *p, *pu;
```

```
double dt, Rho, Cp, Cp_Rho, t;

double alpha = 450; // Heat transfer coefficient

double lambda;  // Thermal conductivity Structural Steel

double TempMax, TempMin;

SplineInterpolation *lambda_table;

approximation *fu_approximate, *fu_approximate1;

double regularization_parameter = 0.0; //  set to zero to begin with Oseen

int main(void)

{

            timerclass T;

            T.ShowTimeBegin();

// Definition of the dimension of space and order of the derivatives

            SetTupleDimensionOrder(2,2);

// Defining the bounding box

            double xmin, ymin, xmax, ymax;

            double xmin_ext, ymin_ext, xmax_ext, ymax_ext;

            xmin = ymin = -1.0;  xmax = ymax = 1.0;

   SetBoundingBox2D( xmin, ymin, xmax, ymax );
```

```
        AdjustBoundingBox2D(xmin, ymin, xmax, ymax, xmin_ext, ymin_ext,
xmax_ext, ymax_ext);

//      Defining information about B-splines: degree, grid

        int degree, nx,ny,  gauss_points, n_random;

        degree = 4;     // degree of B-splines

        nx = ny = 61; // grid nx X ny

        gauss_points = 5; n_random = 5;

        double eps = 0.5;

//  Thermal conductivity (W/m-K) data for Alumina  (Temp in K)

        double Temp[10], K[10];

        int number_of_data_pts = 5;

        Temp[0] = 298.15;    K[0] = 46.0;

        Temp[1] = 400.15;    K[1] = 32.3;

        Temp[2] = 500.15;    K[2] = 24.2;

        Temp[3] = 600.15;    K[3] = 18.9;

        Temp[4] = 800.15;    K[4] = 13.0;

        Rho = 3960; // density kg/m^3   Cp  = 850; // J/kg-K

        lambda_table = new SplineInterpolation(Temp, K, number_of_data_pts);
```

```
            TempMin = Temp[0];  TempMax = Temp[4];

            lambda = K[0];  dt = 1000;  Cp_Rho = 1/(Cp*Rho);

//          Allocation of the matrix and vector

            matrixclass a;

            vectorclass b, C_previous, C_current, C_previousT, C_currentT;

            a.CreateMatrix(M_BANDED, degree, nx, ny);

            b.CreateVector(degree, nx, ny);

//  Assign C for time iteration

C_previousT.CreateVector(degree, nx, ny); C_currentT.CreateVector(degree, nx, ny);

            c_previousT = &C_previousT; c_currentT = &C_currentT;

//          Assign C for Conductivity iteration

C_previous.CreateVector(degree, nx, ny); C_current.CreateVector(degree, nx, ny);

            c_previous = &C_previous; c_current = &C_current;

//  Defining information about basis functions

            tuple_seq P1(1), Pu(1); p = &P1; pu = &Pu;

//  The first sequence is B-splines of degree ns, with grid nx*ny

            p->SetBsplines2DDegree(0, degree, nx, ny); // 0's sequence

            pu->SetBsplines2DDegree(0, degree, nx, ny);
```

```
          quadtreeclass tree(nx, ny, xmin_ext, ymin_ext, xmax_ext, ymax_ext);
// Create quad-tree decomposition of the space using specified number gaussian
integration points;
          tree.BuildTree(gauss_points, eps, n_random, pre);
//  Define which matrix and vector will be assembled by the integration procedures
          SetMatrixPointer( &a );
          SetVectorPointer( &b );
//        Assemble matrix and vector for initial time step
          tree.IntegrateMV(fa_InitialT, fb_initialT); // domain
          ContourIntegrateArcMV(nx, ny, gauss_points, 0.0, 0.0, 0.25, 0.0, PI2,
                                          fa1_initial, fb1_initial ); // boundary
          ContourIntegrateArcMV(nx, ny, gauss_points, 0.0, 0.0, 1.0, 0.0, PI2,
                                          fa1_initial, fb1_initial );
//  Solve linear algebra problem
          a.Solve( b );
          C_currentT = b; // on return b contains values of C_i for first time step
                    fu_approximate = new approximation(degree, nx, ny, gauss_points, eps,
                                          n_random, xmin, ymin, xmax,ymax, fu1, pre);
```

```
C_previous=C_currentT;

do

{

        a.SetZero();

        b.SetZero();

        tree.IntegrateMV(fa, fb_initial);

        ContourIntegrateArcMV(nx, ny, gauss_points, 0.0, 0.0, 0.25, 0.0, PI2,

                                        fa1, fb1 );

        ContourIntegrateArcMV(nx, ny, gauss_points,0.0, 0.0, 1.0, 0.0, PI2,

                                        fa1, fb1 );

        a.Solve( b ); // on return b contains values of C_i

        C_current = b;

        fu_approximate1 = new approximation(degree, nx, ny, gauss_points, eps,

                              n_random, xmin, ymin, xmax,ymax, fu, pre);

        delete fu_approximate;

        fu_approximate = fu_approximate1;

        err = C_current.RelativeErrorEstimation( C_previous, V_NORM_2 );

        C_previous = C_current;
        }while( err > 0.001 ); // Setting convegence criteria


        SetArgumentX(1, x);

           SetArgumentX(2, y);

        filexy( 299,299, xmin,  ymin,  xmax,  ymax, fu, om);

//      Stepping through time

        int k;

        for(k=1;k<30;k++)

        {

                C_previousT = C_current;

                do

                {

                        a.SetZero();

                        b.SetZero();

                        tree.IntegrateMV(fa,fb);

                ContourIntegrateArcMV(nx, ny, gauss_points,0.0, 0.0, 0.25, 0.0, PI2,

                                                  fa1, fb1 );


                ContourIntegrateArcMV(nx, ny, gauss_points,   0.0, 0.0, 1.0, 0.0, PI2,
                                                  fa1, fb1 );
                              a.Solve( b ); // on return b contains values of C_i
                              C_current = b;
                    u_approximate1 = new approximation(degree, nx, ny, gauss_points, eps,
                              n_random, xmin, ymin, xmax,ymax, fu, pre);
                              delete fu_approximate;
                              fu_approximate = fu_approximate1;
                    err = C_current.RelativeErrorEstimation( C_previous, V_NORM_2 );
                              C_previous = C_current;
                    } while( err > 0.001 ); // Setting convegence criteria
                     filexy( 299,299, xmin,  ymin,  xmax,  ymax, fu, om);
}
          T.ShowTime();
          return(0);
}
//Point membership classification function
int pre(void)
{
          if( om() >= 0.0 )
```

```
                return 1;
        else
                return -1;
}
// Initial temperature distribution
double fu_initial(void)
{
        double x, y;
        x = GetArgumentX(1);
        y = GetArgumentX(2);
        return 295.15; // Initial temperature set to zero
}
tuple omega(void)
{
        return circle(0,0,1) & mcircle(0,0,0.25);
}
double om(void)
{
        return omega().GetValue();
```

```
}

// Set temperature surrounding boundaries (T_env can be a function of time)

tuple fi(void)

{

        tuple w1, w2, f1, f2;

        w1 = circle(0,0,1);

        w2 = mcircle(0,0,0.25);

        f1 = 320*alpha; f2 = 650*alpha;

        return paste(w1,w2,f1,f2);

}

// Nonhomogeneous part of solution structure

tuple u1(void)

{

        return - fi() * omega()/lambda;

}

//Homogeneous part of solution structure

void u0(void)

{

        static double h;
```

```
          h = alpha/lambda;

          static tuple om(GetTupleMaxOrder());

          om = omega();

          pu->Compute(0);

          *p = *pu + ((*pu*h) - d1(*pu, om))*om;

          return;

}

// Matrix function for damain integral

double fa_InitialT(int flag, int i, int j)

{

          double result1,result2,result3;

//  Compute basis function at the point

          if( flag == 0 ) // new integration point

          {

                    u0();

          }



//  Computation of the dot product of the gradients

          result1 =   dx_direct( p->GetTuple(0,i), 1)*dx_direct( p->GetTuple(0,j), 1);
```

```
        result2 =   dx_direct( p->GetTuple(0,i), 2)*dx_direct( p->GetTuple(0,j), 2);

        result3 =   dx_direct( p->GetTuple(0,i), 0)*dx_direct( p->GetTuple(0,j), 0);

        return Cp_Rho*dt*(result1 + result2)*lambda + result3;

}

// Matrix fuction for boundary integral

double fa1_initial(int flag, int i, int j)

{

        if( flag == 0 )

        {

                u0();

        }

        return Cp_Rho*dt*alpha* dx_direct( p->GetTuple(0,i), 0 ) * dx_direct( p-
>GetTuple(0,j), 0 );

}

// Initial time step vector function for domain integral

double fb_initialT(int flag, int i)

{

        double result1,result2;
        static double prev;

        static tuple uu1(GetTupleMaxOrder());

        if( flag == 0 )

        {

        u0();

                uu1 = u1();

                prev = fu_initial() - dx_direct( &uu1, 0); ;

        }

        result1 =  dx_direct( p->GetTuple(0,i), 1) * dx_direct( &uu1, 1 );

        result2 =  dx_direct( p->GetTuple(0,i), 2 )*dx_direct( &uu1, 2 );

                return -Cp_Rho*dt*(result1 + result2)*lambda + dx_direct( p-
        >GetTuple(0,i), 0 )    * prev;

}

// Initial Vector function for boundary integral

double fb1_initial(int flag, int i)

{

        static double fi1;

        if( flag == 0 )

        {
                fi1 = fi().GetValue();

                u0();

        }

        return Cp_Rho*dt*fi1*dx_direct( p->GetTuple(0,i), 0 );

}

double fa(int flag, int i, int j)

{

        double result1,result2,result3, result4, result5;

//  Compute basis function at the point

        if( flag == 0 ) // new integration point

        {

                lambda = lambda_interpolate();

                u0();

        }

//  Computation of the dot product of the gradients

        result1 =   dx_direct( p->GetTuple(0,i), 1)*dx_direct( p->GetTuple(0,j), 1);

        result2 =   dx_direct( p->GetTuple(0,i), 2)*dx_direct( p->GetTuple(0,j), 2);

        result3 =   dx_direct( p->GetTuple(0,i), 0)*dx_direct( p->GetTuple(0,j), 0);
        result4 =   dx_direct( p->GetTuple(0,i), 1)*dx_direct(&fu_prev, 1);

        result5 =   dx_direct( p->GetTuple(0,i), 2)*dx_direct( &fu_prev, 2);

        return Cp_Rho*dt*(result1 + result2)*lambda + result3 +

                (

                 Cp_Rho*dt*(result4 + result5)*dirLambda_Tprev * dx_direct( p-
>GetTuple(0,j), 0)

                ) * regularization_parameter;

}

double fa1(int flag, int i, int j)

{

        if( flag == 0 )

        {

                lambda = lambda_interpolate();

                u0();

        }

        return Cp_Rho*dt*alpha* dx_direct( p->GetTuple(0,i), 0 ) * dx_direct( p-
>GetTuple(0,j), 0 );

}
```

```
double fb1(int flag, int i)
{
        static double fi1;
        if( flag == 0 )
        {
                lambda = lambda_interpolate();
                fi1 = fi().GetValue();
                u0();
        }
        return Cp_Rho*dt*fi1*dx_direct( p->GetTuple(0,i), 0 );
}
double fb_initial(int flag, int i)
{
        double result1,result2, result3, result4;
        static double prev,dirLambda_Tprev;;
        static tuple uu1(GetTupleMaxOrder());
        static tuple fu_prev(GetTupleMaxOrder());
        if( flag == 0 )
        {
        lambda = lambda_interpolate();
                u0();
                uu1 = u1();
                dirLambda_Tprev = dirLambda_prev();
                prev = fu_initial() - dx_direct( &uu1, 0);
                fu_prev = sum(c_previous, u1(), *p );
        }


        result1 =  dx_direct( p->GetTuple(0,i), 1) * dx_direct( &uu1, 1 );
        result2 =  dx_direct( p->GetTuple(0,i), 2 )* dx_direct( &uu1, 2 );
        result3 =  dx_direct( p->GetTuple(0,i), 1 )* dx_direct( &fu_prev, 1);
        result4 =  dx_direct( p->GetTuple(0,i), 2 )* dx_direct( &fu_prev, 2);
        return -Cp_Rho*dt*(result1 + result2)*lambda + dx_direct( p->GetTuple(0,i), 0 )
* prev +
                 (
                 - Cp_Rho*dt*(result3 + result4)*dirLambda_Tprev * dx_direct( &uu1, 0)
         + Cp_Rho*dt*(result3 + result4)*dirLambda_Tprev * dx_direct( &fu_prev, 0)
```

```
                    ) * regularization_parameter;
}
// Vector function for domain integral
double fb(int flag, int i)
{
        double result1,result2, result3, result4;
        static tuple uu1(GetTupleMaxOrder());
        static double prev;
        if( flag == 0 )
        {
                t = t - dt; // go to the previous time
                double prev_u1 = u1().GetValue();
                t = t + dt; // return to the current time
                lambda = lambda_interpolate();
                uu1 = u1();
                prev = prev_u1 - uu1.GetValue();
        u0();
                prev = (sum( c_previousT, *p ).GetValue()) + prev;
                dirLambda_Tprev = dirLambda_prev();

                fu_prev = sum(c_previous, u1(), *p );

        }

        result1 =  dx_direct( p->GetTuple(0,i), 1) *dx_direct( &uu1, 1 );

        result2 =  dx_direct( p->GetTuple(0,i), 2 )* dx_direct( &uu1, 2 );

        result3 =  dx_direct( p->GetTuple(0,i), 1 )* dx_direct( &fu_prev, 1);

        result4 =  dx_direct( p->GetTuple(0,i), 2 )*  dx_direct( &fu_prev, 2);

        return -Cp_Rho*dt*(result1 + result2)*lambda + dx_direct( p->GetTuple(0,i), 0 )
* prev +

                  (

                - Cp_Rho*dt*(result3 + result4)*dirLambda_Tprev * dx_direct( &uu1, 0)

         + Cp_Rho*dt*(result3 + result4)*dirLambda_Tprev * dx_direct( &fu_prev, 0)

                  ) * regularization_parameter;

}

// Interpolating thermal conductivity

double lambda_interpolate(void)

{

        double fu_prev;

        fu_prev = fu_approximate->Compute().GetValue();
```

```
            if(fu_prev < TempMin ) fu_prev = TempMin;

            if(fu_prev > TempMax ) fu_prev = TempMax;

            return lambda_table->Compute(fu_prev);

}

// Compute dirivative of lambda with respect to temperature obtained at previous iteration

double dirLambda_prev(void)

{

            double fu_prev;

            u0();

//  Compute the sum

            fu_prev= (sum(c_previous, u1(), *p ).GetValue());

            if(fu_prev < TempMin ) fu_prev = TempMin;

            if(fu_prev > TempMax ) fu_prev = TempMax;

            return lambda_table->ComputeDx1(fu_prev);

}


// SOLUTION
// Solution for Initial Iteration

double fu1(void)

{

        u0();

        return (sum( c_currentT, u1(), *p ).GetValue());

}

// Solution for iteration procedure

double fu(void)

{

        lambda = lambda_interpolate();

        u0();

        return (sum( c_current, u1(), *p ).GetValue());

}
```