Approximations and Object-Oriented Implementation for a Parabolic Partial Differential Equation

R. Chris Camphouse

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Mathematics

John A. Burns, Chair
Eugene M. Cliff
Terry L. Herdman
Jeff Borggaard

January 27, 1999
Blacksburg, Virginia
Approximations and Object-Oriented Implementation for a Parabolic Partial Differential Equation

R. Chris Camphouse

(ABSTRACT)

This work is a numerical study of the 2-D heat equation with Dirichlet boundary conditions over a polygonal domain. The motivation for this study is a chemical vapor deposition (CVD) reactor in which a substrate is heated while being exposed to a gas containing precursor molecules. The interaction between the gas and the substrate results in the deposition of a compound thin film on the substrate.

Two different numerical approximations are implemented to produce numerical solutions describing the conduction of thermal energy in the reactor. The first method used is a Crank-Nicholson finite difference technique which transforms the 2-D heat equation into an algebraic system of equations. For the second method, a semi-discrete method is used which transforms the partial differential equation into a system of ordinary differential equations.

The goal of this work is to investigate the influence of boundary conditions, domain geometry, and initial condition on thermal conduction throughout the reactor. Once insight is gained with respect to the aforementioned conditions, optimal design and control can be investigated. This work represents a first step in our long term goal of developing optimal design and control of such CVD systems.

This work has been funded through Partnerships in Research Excellence and Transition (PRET) grant number F49620-96-1-0329.
In Loving Memory of Mom
Acknowledgements

First and foremost, I would like to thank my friend and research advisor Dr. John Burns. His guidance and support is most appreciated. Allowing freedom to investigate various aspects of this work on my own before discussing my results with him has led to many insights that I may not have discovered otherwise.

Dr. Belinda King has also had an influence on this work. I would like to thank her for convincing me that I have what it takes to go to graduate school. Dr. King also introduced me to my research advisor, Dr. Burns. Both Dr. King and Dr. Burns have opened doors for me that I assumed were closed. I am very appreciative of that.

Thanks are in order for everyone at BEAM Technologies for their hospitality while I was there learning about PDESolve. In particular, Andy Goldenson and Kevin Long have been most helpful. Making themselves available to answer my PDESolve questions via e-mail and telephone calls is very appreciated. Also, Dr. Jeff Borregaard has been indispensable with respect to PDESolve.

Thanks to everyone in the math department and at ICAM for their encouragement and support. Many people from ICAM and the math department have contributed to various aspects of this thesis. Dr. Terry Herdman and Dr. Gene Cliff, both on my advisory committee, have helped with questions that have arisen as this work progressed. Also, my friends Kevin Hulsing and Lisa Stanley deserve a note of thanks for helping me learn how to use Adobe Illustrator. Last, but not least, Melissa Chase of ICAM has been a real joy to work around. Setting up travel arrangements and everything else, always with a smile, has been very helpful. Thanks Melissa!

Finally, I would like to thank my wife and family for all of their love and support. In particular, I would like to thank my wife, Robbie. Allowing me to follow my dreams, and always having the belief that I can achieve them, has not gone unnoticed. Her encouragement and support is what keeps me going.
## Contents

1 **Introduction**  1  
  1.1 Motivation  1  
  1.2 Reactor of Interest  2  
  1.3 Thesis Outline  3  

2 **Model and Approximations**  5  
  2.1 Mathematical Model  5  
  2.2 Initial Boundary Value Problem  9  
  2.3 Approximations on the Rectangle  10  
  2.4 Finite Difference Approximations  11  
    2.4.1 Boundary Conditions  15  
    2.4.2 Matrix Equations  16  
  2.5 Semi-Discrete Approximation  20  
    2.5.1 Boundary Conditions  20  
    2.5.2 Matrix Equations  21  

3 **PDESolve**  24  
  3.1 PDESolve  24  
  3.2 Creating the Domain  25  
  3.3 Partitioning the Domain  27  
  3.4 Spatial Derivatives  28  
  3.5 Boundary Conditions  29  
  3.6 Solving the System  32
4 Numerical Experiments 33
  4.1 Numerical Tests .................................................. 33
    4.1.1 Numerical Tests on the Square ............................ 34
    4.1.2 Numerical Tests on the Entire Domain ...................... 39
  4.2 Geometry Experiments ........................................... 44
  4.3 Numerical Experiments with the Lamps ......................... 51

5 Conclusions 61
  5.1 Overview of Results ............................................ 61
  5.2 Conclusions .................................................... 63
  5.3 Future Work ................................................... 63

A PDEsolve Source Codes 67
  A.1 PDEsolve implementation of the Crank-Nicholson scheme on the polygonal domain ........................................... 67

B Matlab Source Codes 72
  B.1 Codes to construct and solve the semi-discrete approximation on the square 72
# List of Figures

1.1 Chemical Vapor Deposition Reactor ........................................... 2

2.1 Computational Domain $\Omega$ .................................................. 6

2.2 Lamps along $\Gamma_C$ ............................................................ 7

2.3 Temperature Profile Function .................................................. 8

2.4 Spatial Grid on the Square ..................................................... 10

3.1 Domain Composed of Five Regions ......................................... 26

3.2 Tags Used For LinkAcross ..................................................... 31

4.1 Exact and Computed Solutions on the Square with $\Delta t = 0.2$ .... 36

4.2 Exact and Computed Solutions on the Square with $\Delta t = 0.1$ .... 37

4.3 Exact and Computed Solutions on the Square with $\Delta t = 0.05$ .... 38

4.4 Exact and Computed Solution on the Entire Domain with $\Delta t = 0.2$ .... 40

4.5 Exact and Computed Solution on the Entire Domain with $\Delta t = 0.1$ .... 41

4.6 Exact and Computed Solution on the Entire Domain with $\Delta t = 0.05$ .... 42

4.7 Geometry Experiment 1 at $t=0$. (Initial Condition) .................. 45

4.8 Geometry Experiment 1 at $t=2$. ............................................ 45

4.9 Geometry Experiment 1 at $t=4$. ............................................ 46

4.10 Geometry Experiment 1 at $t=8$. ......................................... 46

4.11 Geometry Experiment 1 at $t=40.6$. ...................................... 47

4.12 Geometry Experiment 2 at $t=0$. (Initial Condition) .................. 48

4.13 Geometry Experiment 2 at $t=2$. ............................................ 48

4.14 Geometry Experiment 2 at $t=4$. ......................................... 49

4.15 Geometry Experiment 2 at $t=8$. ......................................... 49
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.16</td>
<td>Geometry Experiment 2 at t=50.2.</td>
<td>50</td>
</tr>
<tr>
<td>4.17</td>
<td>A Sample $u_k(t)$</td>
<td>51</td>
</tr>
<tr>
<td>4.18</td>
<td>Numerical Experiment 1 at t=0. (Initial Condition)</td>
<td>53</td>
</tr>
<tr>
<td>4.19</td>
<td>Numerical Experiment 1 at t=2.</td>
<td>53</td>
</tr>
<tr>
<td>4.20</td>
<td>Numerical Experiment 1 at t=4.</td>
<td>54</td>
</tr>
<tr>
<td>4.21</td>
<td>Numerical Experiment 1 at t=8.</td>
<td>54</td>
</tr>
<tr>
<td>4.22</td>
<td>Numerical Experiment 2 at t=0. (Initial Condition)</td>
<td>55</td>
</tr>
<tr>
<td>4.23</td>
<td>Numerical Experiment 2 at t=2.</td>
<td>55</td>
</tr>
<tr>
<td>4.24</td>
<td>Numerical Experiment 2 at t=4.</td>
<td>56</td>
</tr>
<tr>
<td>4.25</td>
<td>Numerical Experiment 2 at t=8.</td>
<td>56</td>
</tr>
<tr>
<td>4.26</td>
<td>Numerical Experiment 3 at t=0. (Initial Condition)</td>
<td>57</td>
</tr>
<tr>
<td>4.27</td>
<td>Numerical Experiment 3 at t=2.</td>
<td>57</td>
</tr>
<tr>
<td>4.28</td>
<td>Numerical Experiment 3 at t=4.</td>
<td>58</td>
</tr>
<tr>
<td>4.29</td>
<td>Numerical Experiment 3 at t=8.</td>
<td>58</td>
</tr>
<tr>
<td>4.30</td>
<td>Numerical Experiment 4 at t=0. (Initial Condition)</td>
<td>59</td>
</tr>
<tr>
<td>4.31</td>
<td>Numerical Experiment 4 at t=2.</td>
<td>59</td>
</tr>
<tr>
<td>4.32</td>
<td>Numerical Experiment 4 at t=4.</td>
<td>60</td>
</tr>
<tr>
<td>4.33</td>
<td>Numerical Experiment 4 at t=8.</td>
<td>60</td>
</tr>
</tbody>
</table>
## List of Tables

4.1 Errors on the Square ......................................................... 39
4.2 Errors on the Entire Domain ............................................... 43
4.3 Values Used in Geometry Experiments ................................. 44
4.4 Values Used in Numerical Experiments ............................... 52
Chapter 1

Introduction

1.1 Motivation

The deposition of a compound thin film on a desired substrate has many useful applications. Chemical vapor deposition (CVD) is used in the development of silicon wafers and other technologically significant electronic components. In a CVD reactor, a heated substrate is exposed to a gas flow containing precursor molecules. This results in the deposition of a thin film of a desired compound on the substrate. Properties of the thin film are affected significantly by temperature fluctuations during film growth. Thus, being able to influence the temperature of the substrate in the CVD reactor is important. Keeping the substrate at uniform temperature during the deposition of the thin film ensures that the required properties of the film are not destroyed by temperature fluctuations [3]. In many CVD reactors, this is done by placing heating elements outside of the reactor. However, it is not at all obvious how the location, intensity, duration of influence, and other aspects of the heating elements may influence the temperature of the substrate inside the reactor. As a result, it is necessary to construct a mathematical model of the reactor in order to accurately predict the way in which an individual heating element influences the temperature of the substrate during film growth. A first step in creating a complete model is developing a model describing the conduction of thermal energy throughout the reactor. In the physical system, the temperature inside the reactor is altered through heat transfer by way of radiation. In
this work, we will be concerned with conduction only. Thus, the model developed in this
work will not fully model a CVD reactor. However, the model presented here will allow for
variations in reactor geometry. Creating a model for which numerical experiments can be
performed for different reactor geometries will provide insight into the best geometry of the
reactor. The geometry of the reactor, as well as the heating elements placed outside of the
reactor, play a significant role in the temperature of the substrate during film growth.

1.2 Reactor of Interest

Different types of CVD processes are implemented in order to attain the deposition of
different compound thin films [4]. The focus of this work is not to investigate the many
different types of reactors used to develop different thin films, but rather to focus attention
on a problem where the geometry is based on a particular CVD reactor.

![Chemical Vapor Deposition Reactor](image)

Figure 1.1: Chemical Vapor Deposition Reactor

Although energy transfer in this CVD reactor is primarily through radiation, we shall
focus on thermal conduction. Thus, the results in this thesis will not have direct application to the CVD reactor in [3]. The goal of this work is to develop a software package based on PDESolve and to study the feasibility of using such tools in future reactor design and control.

In the reactor of concern in this work, gas-phase precursor molecules in a carrier gas enter through a showerhead located above a magnesium oxide ($MgO$) substrate. The substrate is held in a silicone carbide ($SiC$) susceptor. The carbide susceptor sits above an inconel can containing heating lamps. The inconel can isolates the heating lamps from the oxidizing environment of the reactor. As the gas passes over the substrate, a thin film of superconducting YBCO ($YBa_2Cu_3O_{7-x}$) develops on the substrate. By allowing the lamps to get hotter or colder, the temperature of the substrate can be varied which will, as a result, influence the properties of the YBCO thin film [2, 3].

1.3 Thesis Outline

In Chapter 2, we derive a model of thermal conduction for a geometry defined by the CVD reactor. The model takes the form of a parabolic partial differential equation with Dirichlet boundary conditions. The geometry of the reactor is incorporated into the resulting initial boundary value problem (IBVP). We consider two numerical techniques. The first technique is a finite difference method, commonly referred to as a Crank-Nicholson scheme. A theoretical result guaranteeing the stability of this finite difference scheme is presented. The second scheme is a semi-discrete method based on the method of lines. A discussion of the implementation of boundary conditions for the two different methods is presented.

In Chapter 3, we present an overview of the software package PDESolve. We illustrate how the differential operators, boundary conditions, computational domains, and finite difference meshes can be implemented in PDESolve. In particular, these objects are created in PDESolve to form the matrix equation resulting from the finite difference approximation of the thermal conduction model associated with the CVD reactor. The commands needed to solve the linear system of equations are also presented.
Numerical comparisons of the two numerical schemes are given in Chapter 4. An exact solution to the IBVP is chosen, and the numerical solutions obtained from both approximation methods are compared to the exact solution. This is done on a simple square domain as well as a polygonal computational domain representing the more complex geometry of the reactor. Numerical experiments are presented in order to ascertain the significance of step size on the accuracy of the computed solution. The finite difference method is implemented for different domain geometries and boundary conditions. Numerical results are presented to illustrate the importance of domain geometry, boundary conditions, and initial conditions on thermal conduction throughout the domain.

An overview of results, as well as a discussion of future work, is presented in Chapter 5. Conclusions derived from the results presented in this work are stated.
Chapter 2

Model and Approximations

2.1 Mathematical Model

In order to do numerical experiments to determine the affect of domain geometry, boundary conditions and initial conditions on the conduction of thermal energy throughout the reactor, it is necessary to form a mathematical model for the reactor.

Numerical experiments can then be conducted on the mathematical model.

Since the reactor has a specific geometry, it is desirable to conduct numerical experiments on a computational domain with a similar geometry. By allowing the parameters which specify the shape of the computational domain to be changed, it is possible to observe the influence of different geometries on thermal conduction. For this reason, the computational domain $\Omega$ will be configured as in Figure 2.1. On the boundary $\partial\Omega$, define

$$\Gamma_C = \{(x, b_2)|a_2 \leq x \leq a_3\} \quad \text{(in red)},$$

$$\Gamma_U = \partial\Omega \setminus \Gamma_C.$$ 

Notice that in Figure 2.1, the parameters $a_1$, $a_2$, $a_3$, $a_4$, $b_1$, $b_2$, and $b_3$ completely specify the geometry of the computational domain. By varying these parameters, numerical results will be obtained for different reactor geometries.
Since heating lamps are placed along the boundary of the physical reactor in order to control the temperature, a similar situation needs to occur along the boundary of the computational domain. That is, a time-varying boundary condition is imposed along $\Gamma_C$. As in the physical system, “lamps” are placed along $\Gamma_C$ in the computational problem.

Associated with each lamp is a temperature profile function $b_k(x)$ which represents the influence of the $k$-th lamp on $\Gamma_C$. In the work presented here, it is assumed that the lamps are close enough to $\Gamma_C$ so that their individual influences do not overlap. In addition, it is assumed that each lamp will have the greatest impact on the portion of $\Gamma_C$ directly above it. Thus, each lamp has the greatest impact on the portion of $\Gamma_C$ directly above its center. With these assumptions, the temperature profile function associated with each lamp will attain its maximum value in the center of the lamp. Since the influences of the lamps are assumed to be nonintersecting, the temperature profile is negligible at the extreme right and left of each lamp. It is also assumed that each lamp is cold when it is off. That is, the portion of $\Gamma_C$ directly above a lamp which is off will be held at room temperature until the lamp turns
Con. In the numerical results that are presented, the temperature profiles are constructed as portions of sine functions.

With $N$ representing the number of lamps along $\Gamma_C$ and $\text{heatmax}_k$ representing the maximum allowable temperature of the $k$-th lamp, the $k$-th temperature profile function $b_k(x)$ is specified as

$$b_k(x) = \begin{cases} \text{heatmax}_k \left| \sin \left( \frac{N \pi (x - a_2)}{a_3 - a_2} \right) \right|, & a_2 + \frac{(k-1)(a_3-a_2)}{N} \leq x \leq a_2 + \frac{k(a_3-a_2)}{N}; \\ 0, & \text{otherwise}. \end{cases} \quad (2.3)$$

Specifying $T_0$ to equal room temperature, temperatures throughout $\Omega$ are normalized in such a way as to allow a value of zero for room temperature. Representing the temperature throughout the domain by $T(t, x, y)$ and setting $T(t, x, y) = [T(t, x, y) - T_0]$ normalizes room temperature to zero.

By multiplying $b_k(x)$ by a scalar function $u_k(t)$, the contribution of a particular lamp to the boundary condition along $\Gamma_C$ can be represented as $b_k(x)u_k(t)$. In this way, it is possible to control the influence of the individual lamps on the boundary by adjusting the
functions $u_k(t)$. For example, if $u_3(t)$ is identically zero, then the 3rd lamp is “off”. If $u_3(t)$ is identically equal to one, the 3rd lamp is heating as much as is possible and the temperature of the lamp is being held at its maximum possible value.

Since the contribution of each lamp to the boundary condition along $\Gamma_C$ is given by $b_k(x)u_k(t)$, the total contribution of all of the lamps to the boundary condition at $\Gamma_C$ is written as $\sum_{k=1}^{N} b_k(x)u_k(t)$ where $N$ is the number of lamps, $b_k(x)$ is the temperature profile function for the $k$-th lamp, and $u_k(t)$ is a scalar function allowing for control of the $k$-th lamp. In the physical reactor, the remaining boundary is held at room temperature as time evolves. Therefore, in the computational problem, the temperature along $\Gamma_U$ will be held fixed at a normalized value of zero as time evolves.
2.2 Initial Boundary Value Problem

The model discussed thus far leads to a initial boundary value problem describing thermal conduction throughout the reactor. The partial differential equation modeling the changing temperature $T(t, x, y)$ in a two-dimensional domain is the 2-D heat equation

$$\frac{\partial}{\partial t}T(t, x, y) = K \left( \frac{\partial^2}{\partial x^2}T(t, x, y) + \frac{\partial^2}{\partial y^2}T(t, x, y) \right), \quad t > 0, \quad (x, y) \in \Omega. \quad (2.4)$$

In (2.4), $K \geq 0$ is a thermal diffusivity constant. The initial and boundary values that must be imposed in order to model the temperature throughout the reactor are

$$T(0, x, y) = f(x, y), \quad (2.5)$$

$$T(t, x, y) \bigg|_{\Gamma_C} = \sum_{k=1}^{N} b_k(x)u_k(t), \quad t > 0, \quad (2.6)$$

$$T(t, x, y) \bigg|_{\Gamma_U} = 0, \quad t > 0. \quad (2.7)$$

In condition (2.5), $f(x, y)$ is the temperature distribution throughout the domain at $t=0$. Note that setting each $u_k(t)$ identically equal to zero in boundary condition (2.6) leads to the 2-D heat equation with the entire boundary $\partial \Omega$ being held at zero temperature as time evolves. Numerical experiments on different reactor geometries with the entire boundary held at zero temperature will provide insight into the importance of reactor geometry.
2.3 Approximations on the Rectangle

Since the geometry for the physical system is somewhat complicated, difference approximations to the two-dimensional heat equation are constructed on the rectangle $\Omega_0 = [a_2, a_3] \times [b_2, b_3]$. The boundary conditions imposed on the rectangle are similar to those imposed on the more complicated domain. With $\Gamma_C = \{(x, b_2) | a_2 \leq x \leq a_3\}$ as before and $\Gamma_{U_0} = \partial \Omega_0 \setminus \Gamma_C$, the boundary conditions specified on the simpler computational domain are identical to the ones specified on the more complicated domain. That is,

\begin{equation}
T(0, x, y) = f(x, y),
\end{equation}

\begin{equation}
T(t, x, y) \bigg|_{\Gamma_C} = \sum_{k=1}^{N} b_k(x) u_k(t), \quad t > 0,
\end{equation}

\begin{equation}
T(t, x, y) \bigg|_{\Gamma_{U_0}} = 0, \quad t > 0.
\end{equation}

A finite difference approximation and a semi-discrete approximation to the initial boundary value problem are constructed on the simpler computational domain.

Figure 2.4: Spatial Grid on the Square

Subdivide $[a_2, a_3]$ into subintervals of length $\Delta x$ starting from $x_0 = a_2$ and ending with $x_{p+1} = a_3$. Similarly, partition $[b_2, b_3]$ into subintervals of length $\Delta y$ starting from $y_0 = b_2$
and ending with \( y_{q+1} = b3 \). This results in \( p \) evaluation points in the \( x \) direction and \( q \) evaluation points in the \( y \) direction. In the work presented here, \( \Delta y = \Delta x = h \).

Difference approximations to the spatial and time derivatives are constructed on the partitioned domain. Note that \((x_0, y_j)\) and \((x_{p+1}, y_j)\) are grid points on the lines \( x = a2 \) and \( x = a3 \), respectively. In a similar fashion, \((x_i, y_0)\) and \((x_i, y_{q+1})\) are grid points on the lines \( y = b2 \) and \( y = b3 \). Thus, these grid points are on the boundary of \( \Omega_0 \). As a result, the Dirichlet boundary conditions imposed by the initial boundary value problem are imposed on these grid points.

The grid pictured in Figure 2.4 as well as the difference approximations in space and time, where appropriate, will yield a matrix equation for both the finite difference approximation and the semi-discrete approximation.

### 2.4 Finite Difference Approximations

To numerically approximate the solution of the problem on the square computational domain, a finite difference approximation to the partial differential equation is constructed. This requires the discretization of the time and spatial derivatives. Time discretizations must be done with care. If one begins with the obvious explicit scheme and uses a time-step of \( \Delta t \), spatial step-sizes of \( \Delta x \) and \( \Delta y \), the first derivative in time is approximated by the forward difference

\[
\frac{\partial}{\partial t} T(t, x, y) \approx \frac{T(t + \Delta t, x, y) - T(t, x, y)}{\Delta t}.
\] (2.11)

In a similar fashion, the second order spatial derivatives in the \( x \) and \( y \) directions are constructed as difference approximations of the form

\[
\frac{\partial^2}{\partial x^2} T(t, x, y) \approx \frac{T(t, x + \Delta x, y) - 2T(t, x, y) + T(t, x - \Delta x, y)}{\Delta x^2},
\] (2.12)

\[
\frac{\partial^2}{\partial y^2} T(t, x, y) \approx \frac{T(t, x, y + \Delta y) - 2T(t, x, y) + T(t, x, y - \Delta y)}{\Delta y^2}.
\] (2.13)
In order to write the difference approximations (2.11)-(2.13) in a concise manner, it is necessary to introduce some notation. Evaluation points in the \( x \) and \( y \) directions, as well as specific values if time, can be specified as

\[
t_n = (n - 1)\Delta t,
\]

\[
x_i = a2 + (i - 1)\Delta x,
\]

\[
y_j = b2 + (j - 1)\Delta y.
\]

Using the notation of (2.14)-(2.16), represent the function evaluations in (2.11)-(2.13) at each evaluation point \((x_i, y_j)\) as

\[
T_{i,j}^n = T(t_n, x_i, y_j).
\]

Then, the discretizations (2.11), (2.12), and (2.13) at each evaluation point \((x_i, y_j)\) are written utilizing the notation of (2.17) as

\[
\frac{\partial}{\partial t} T(t, x_i, y_j) \approx \frac{T_{i,j}^{n+1} - T_{i,j}^n}{\Delta t},
\]

\[
\frac{\partial^2}{\partial x^2} T(t, x_i, y_j) \approx \frac{T_{i+1,j}^n - 2T_{i,j}^n + T_{i-1,j}^n}{\Delta x^2},
\]

\[
\frac{\partial^2}{\partial y^2} T(t, x_i, y_j) \approx \frac{T_{i,j+1}^n - 2T_{i,j}^n + T_{i,j-1}^n}{\Delta y^2}.
\]

Substituting approximations (2.18) - (2.20) into the 2-D heat equation (2.4) leads to a finite difference approximation at each evaluation point on the grid of the form

\[
\frac{T_{i,j}^{n+1} - T_{i,j}^n}{\Delta t} = K \left( \frac{T_{i+1,j}^n - 2T_{i,j}^n + T_{i-1,j}^n}{\Delta x^2} + \frac{T_{i,j+1}^n - 2T_{i,j}^n + T_{i,j-1}^n}{\Delta y^2} \right).
\]

The finite difference approximation (2.21) is an explicit scheme. In other words, the solution at time-step \( n + 1 \) is written explicitly as a function of the solution at time-step \( n \). In order for the difference approximation to give an accurate numerical solution, it is necessary that errors introduced in the approximation do not increase as the number of
iterations increases. A scheme in which the errors introduced by the approximation do not grow with the number of iterations, regardless of the size of the time-step, is classified as\textit{unconditionally stable}. A numerical scheme which restricts the time-step size in order to disallow errors from growing with the number of iterations is classified as\textit{conditionally stable} \cite{10}. It has been shown that finite difference scheme (2.21) imposes a severe restriction on the time-step $\Delta t$ in order to guarantee numerical stability. The following result may be found in \cite{6}.

\begin{theorem}[Stability of the Explicit Scheme]
The numerical approximation to the 2-D heat equation obtained from discretization (2.21) is conditionally stable where the time-step $\Delta t$ must satisfy

$$K \left( \frac{\Delta t}{(\Delta x)^2} + \frac{\Delta t}{(\Delta y)^2} \right) \leq \frac{1}{2}.$$ 

\end{theorem}

The restriction imposed on the time-step size by Theorem 2.4.1 is impractical. If $\Delta x$ and $\Delta y$ are taken to be small in order to yield a sufficiently accurate numerical solution in space, then the time-step $\Delta t$ must be taken extremely small in order to guarantee numerical stability. For this reason, the fully explicit scheme has the potential of being very costly. A better finite difference scheme is needed.

The approximations (2.19) and (2.20) to the derivatives $\frac{\partial^2}{\partial x^2} T(t, x, y)$ and $\frac{\partial^2}{\partial y^2} T(t, x, y)$ can be evaluated at the $n + 1$ time-step. Doing so results in difference approximations to the spatial derivatives at time-step $n + 1$ of the form

$$\frac{\partial^2}{\partial x^2} T(t + \Delta t, x_i, y_j) \approx \frac{T_{i+1,j}^{n+1} - 2T_{i,j}^{n+1} + T_{i-1,j}^{n+1}}{\Delta x^2},$$

$$\frac{\partial^2}{\partial y^2} T(t + \Delta t, x_i, y_j) \approx \frac{T_{i,j+1}^{n+1} - 2T_{i,j}^{n+1} + T_{i,j-1}^{n+1}}{\Delta y^2}.\quad (2.22)$$

Using these approximations for the spatial derivatives and approximation (2.11) for the derivative in time results in an implicit approximation to the 2-D heat equation at each
grid point \((x_i, y_j)\). The solution at time-step \(n + 1\) is written implicitly as a function of the solution at time-step \(n + 1\) as

\[
\frac{T_{i,j}^{n+1} - T_{i,j}^n}{\Delta t} = K \left( \frac{T_{i+1,j}^{n+1} - 2T_{i,j}^{n+1} + T_{i-1,j}^{n+1}}{\Delta x^2} + \frac{T_{i,j+1}^{n+1} - 2T_{i,j}^{n+1} + T_{i,j-1}^{n+1}}{\Delta y^2} \right).
\] (2.24)

Forming the average of the explicit scheme (2.21) and the implicit scheme (2.24) results in the familiar Crank-Nicholson scheme

\[
\frac{T_{i,j}^{n+1} - T_{i,j}^n}{\Delta t} = K \left( \frac{T_{i+1,j}^n - 2T_{i,j}^n + T_{i-1,j}^n}{\Delta x^2} + \frac{T_{i,j+1}^n - 2T_{i,j}^n + T_{i,j-1}^n}{\Delta y^2} \right).
\] (2.25)

Again, one finds the following result in [6].

THEOREM 2.4.2 (STABILITY OF THE CRANK-NICHOLSON SCHEME)

The semi-implicit Crank-Nicholson finite difference scheme (2.25) is unconditionally stable.

The Crank-Nicholson scheme can be simplified for computations in which a uniform mesh is used. For equal spatial step sizes \(\Delta x = h = \Delta y\), equation (2.25) simplifies to

\[
\frac{T_{i,j}^{n+1} - T_{i,j}^n}{\Delta t} = K \left( \frac{T_{i+1,j}^n - 2T_{i,j}^n + T_{i-1,j}^n}{2h^2} + \frac{T_{i,j+1}^n - 2T_{i,j}^n + T_{i,j-1}^n}{2h^2} \right)
\] (2.26)

Setting \(\nu = \frac{K\Delta t}{2h^2}\) and grouping terms at time-step \(n + 1\) on the left and time-step \(n\) on the right leads to

\[
T_{i,j}^{n+1} = T_{i,j}^n + \nu(T_{i+1,j}^n - 4T_{i,j}^n + T_{i-1,j}^n + T_{i,j+1}^n + T_{i,j-1}^n) \quad (2.27)
\]
Factoring $\nu$ and collecting like terms yields the final simplification of the Crank-Nicholson scheme given by

$$
\left\{ \left( \frac{1+4\nu}{\nu} \right) T_{i,j}^{n+1} - T_{i+1,j}^{n+1} - T_{i-1,j}^{n+1} \right\} - \left\{ T_{i,j}^{n+1} \right\} - \left\{ T_{i,j-1}^{n+1} \right\} = \left\{ \left( \frac{1-4\nu}{\nu} \right) T_{i,j}^{n} + T_{i+1,j}^{n} + T_{i-1,j}^{n} \right\} + \left\{ T_{i,j+1}^{n} \right\} + \left\{ T_{i,j-1}^{n} \right\}.
$$

(2.28)

### 2.4.1 Boundary Conditions

Specific values of $i$ and $j$ in (2.28) correspond to the boundary of the computational domain. Boundary conditions are imposed on the boundary of $\Omega_0$ by equations (2.9)-(2.10). Thus, evaluation points on the boundary of the computational domain need to take specific values in order to agree with the boundary conditions specified by the initial boundary value problem.

If $i = 1$, then $T_{i-1,j}^{n+1} = T_{0,j}^{n+1}$ and $T_{i-1,j}^{n} = T_{0,j}^{n}$. Thus, the value of $i = 1$ corresponds to grid points along the line $x = a_2$. Similarly, if $i = p$, then $T_{i+1,1}^{n+1} = T_{p+1,1}^{n}$ and $T_{i+1,1}^{n} = T_{p+1,1}^{n}$. As a result, the value $i = p$ corresponds to grid points along the line $x = a_3$. The conditions $T(t, a_2, y) = 0$ and $T(t, a_3, y) = 0$ are imposed by the boundary conditions of the 2-D heat equation for all values of $t$. As a result, the temperature at grid points along the lines $x = a_2$ and $x = a_3$ is held fixed at zero as time evolves. Thus,

$$
T_{0,j}^{n+1} = 0, \quad T_{0,j}^{n} = 0,
$$

(2.29)

$$
T_{p+1,j}^{n+1} = 0, \quad \text{and} \quad T_{p+1,j}^{n} = 0.
$$

(2.30)

If $j = 1$, then $T_{i,j-1}^{n+1} = T_{i,0}^{n+1}$ and $T_{i,j-1}^{n} = T_{i,0}^{n}$. Therefore, this value of $j$ corresponds to grid points along $y = \Gamma_C$. From the initial boundary value problem discussed previously, a time varying boundary condition is imposed along $\Gamma_C$ of the form

$$
T(t, x, y) \bigg|_{\Gamma_C} = \sum_{k=1}^{N} b_k(x) u_k(t).
$$
Thus, in order for the finite difference scheme to agree with the boundary condition along $\Gamma_C$, it is necessary that the values

$$T^{n+1}_{i,0} = \sum_{k=1}^{N} b_k(x_i) u_k(t + \Delta t), \quad (2.31)$$

$$T^n_{i,0} = \sum_{k=1}^{N} b_k(x_i) u_k(t) \quad (2.32)$$

be given to evaluation points corresponding to $\Gamma_C$.

Finally, if $j = q$, then $T^{n+1}_{i,j+1} = T^{n+1}_{i,q+1}$ and $T^n_{i,j+1} = T^n_{i,q+1}$. As a result, this value of $j$ corresponds to grid points along the line $y = b3$. The boundary conditions that must be imposed on the temperature at these grid points are given by

$$T^{n+1}_{i,q+1} = 0 \quad \text{and} \quad T^n_{i,q+1} = 0. \quad (2.33)$$

### 2.4.2 Matrix Equations

Allowing $i$ to vary between 1 and $p$ for each value of $j$ in difference equation (2.28) results in a matrix equation for each $1 \leq j \leq q$.

Let

$$H = \begin{bmatrix}
\frac{1+4\nu}{\nu} & -1 & 0 & \ldots & 0 & 0 & 0 \\
-1 & \frac{1+4\nu}{\nu} & -1 & \ldots & 0 & 0 & 0 \\
0 & -1 & \frac{1+4\nu}{\nu} & \ldots & 0 & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & 0 & \ldots & \frac{1+4\nu}{\nu} & -1 & 0 \\
0 & 0 & 0 & \ldots & -1 & \frac{1+4\nu}{\nu} & -1 \\
0 & 0 & 0 & \ldots & 0 & -1 & \frac{1+4\nu}{\nu}
\end{bmatrix}_{p \times p}$$

(2.34)
and

\[
S = \begin{bmatrix}
\frac{1-4\nu}{\nu} & 1 & 0 & \ldots & 0 & 0 & 0 \\
1 & \frac{1-4\nu}{\nu} & 1 & \ldots & 0 & 0 & 0 \\
0 & 1 & \frac{1-4\nu}{\nu} & \ldots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & \frac{1-4\nu}{\nu} & 1 & 0 \\
0 & 0 & 0 & \ldots & 1 & \frac{1-4\nu}{\nu} & 1 \\
0 & 0 & 0 & \ldots & 0 & 1 & \frac{1-4\nu}{\nu}
\end{bmatrix}_{p \times p}
\]  \quad (2.35)

Also, define the vectors

\[
T_j^{n+1} = \begin{bmatrix}
T_1^{n+1} \\
T_2^{n+1} \\
T_3^{n+1} \\
\vdots \\
T_p^{n+1}
\end{bmatrix}_{p \times 1}, \quad T_j^n = \begin{bmatrix}
T_1^n \\
T_2^n \\
T_3^n \\
\vdots \\
T_p^n
\end{bmatrix}_{p \times 1}, \quad B_i = \begin{bmatrix}
b_i(x_1) \\
b_i(x_2) \\
b_i(x_3) \\
\vdots \\
b_i(x_p)
\end{bmatrix}_{p \times 1}
\]  \quad (2.36)

If \( I \) denotes the \( p \times p \) identity matrix, then the Crank-Nicholson scheme (2.28) can be written as

\[
HT_j^{n+1} - IT_j^{n+1} - IT_j^{n+1} = ST_j^n + IT_j^n + IT_j^n.
\]  \quad (2.37)

If \( j = 1 \), then \( T_j^{n+1} = T_0^{n+1} \) and \( T_j^n = T_0^n \). Implementing the value given by (2.31) on
these grid points at time-step $n + 1$ yields a vector of sums of the form

$$
\begin{bmatrix}
T_{1,0}^{n+1} \\
T_{2,0}^{n+1} \\
T_{3,0}^{n+1} \\
\vdots \\
T_{p-1,0}^{n+1} \\
T_{p,0}^{n+1}
\end{bmatrix}_{p \times 1} =
\begin{bmatrix}
\sum_{k=1}^{N} b_k(x_1) u_k(t + \Delta t) \\
\sum_{k=1}^{N} b_k(x_2) u_k(t + \Delta t) \\
\sum_{k=1}^{N} b_k(x_3) u_k(t + \Delta t) \\
\vdots \\
\sum_{k=1}^{N} b_k(x_{p-2}) u_k(t + \Delta t) \\
\sum_{k=1}^{N} b_k(x_{p-1}) u_k(t + \Delta t) \\
\sum_{k=1}^{N} b_k(x_p) u_k(t + \Delta t)
\end{bmatrix}_{p \times N}
\begin{bmatrix}
b_1(x_1) & b_2(x_1) & \cdots & b_N(x_1) \\
b_1(x_2) & b_2(x_2) & \cdots & b_N(x_2) \\
b_1(x_3) & b_2(x_3) & \cdots & b_N(x_3) \\
\vdots & \vdots & \vdots & \vdots \\
b_1(x_{p-2}) & b_2(x_{p-2}) & \cdots & b_N(x_{p-2}) \\
b_1(x_{p-1}) & b_2(x_{p-1}) & \cdots & b_N(x_{p-1}) \\
b_1(x_{p}) & b_2(x_{p}) & \cdots & b_N(x_{p})
\end{bmatrix}_{N \times N}
$$

(2.38)

The vector of sums in (2.38) can be written as the product of a matrix and a vector. Forming a $p \times N$ matrix whose $(i, j)$th component is $b_j(x_i)$ and a $N \times 1$ vector whose $i$-th row is the scalar function $u_i(t + \Delta t)$, (2.38) is written as

$$
\begin{bmatrix}
\sum_{k=1}^{N} b_k(x_1) u_k(t + \Delta t) \\
\sum_{k=1}^{N} b_k(x_2) u_k(t + \Delta t) \\
\sum_{k=1}^{N} b_k(x_3) u_k(t + \Delta t) \\
\vdots \\
\sum_{k=1}^{N} b_k(x_{p-2}) u_k(t + \Delta t) \\
\sum_{k=1}^{N} b_k(x_{p-1}) u_k(t + \Delta t) \\
\sum_{k=1}^{N} b_k(x_p) u_k(t + \Delta t)
\end{bmatrix}_{p \times 1} =
\begin{bmatrix}
b_1(x_1) & b_2(x_1) & \cdots & b_N(x_1) \\
b_1(x_2) & b_2(x_2) & \cdots & b_N(x_2) \\
b_1(x_3) & b_2(x_3) & \cdots & b_N(x_3) \\
\vdots & \vdots & \vdots & \vdots \\
b_1(x_{p-2}) & b_2(x_{p-2}) & \cdots & b_N(x_{p-2}) \\
b_1(x_{p-1}) & b_2(x_{p-1}) & \cdots & b_N(x_{p-1}) \\
b_1(x_{p}) & b_2(x_{p}) & \cdots & b_N(x_{p})
\end{bmatrix}_{N \times N}
\begin{bmatrix}
u_1(t + \Delta t) \\
u_2(t + \Delta t) \\
u_3(t + \Delta t) \\
\vdots \\
u_{N-2}(t + \Delta t) \\
u_{N-1}(t + \Delta t) \\
u_N(t + \Delta t)
\end{bmatrix}_{N \times 1}
$$

(2.39)

Implementing the values prescribed by (2.32) for grid points comprising $\mathbf{T}_0^m$ results in a
similar matrix product at time-step \( n \), written as

\[
\begin{bmatrix}
T^n_{1,0} & b_1(x_1) & b_2(x_1) & \cdots & b_N(x_1) \\
T^n_{2,0} & b_1(x_2) & b_2(x_2) & \cdots & b_N(x_2) \\
T^n_{3,0} & b_1(x_3) & b_2(x_3) & \cdots & b_N(x_3) \\
\vdots & \ddots & \ddots & \ddots & \ddots \\
T^n_{p,0} & b_1(x_p) & b_2(x_p) & \cdots & b_N(x_p)
\end{bmatrix}
\begin{bmatrix}
T^{n+1}_1 \\
T^{n+1}_2 \\
\vdots \\
T^{n+1}_q \\
p \times N
\end{bmatrix}
\begin{bmatrix}
B_1 & B_2 & \cdots & B_{N-1} & B_N \\
0 & 0 & \cdots & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & \cdots & 0 & 0
\end{bmatrix}
\begin{bmatrix}
u_1(t) \\
u_2(t) \\
\vdots \\
u_N(t)\\N \times 1
\end{bmatrix}
\]

(2.40)

With the boundary values prescribed by the IBVP incorporated into the finite difference scheme, it is now possible to write the system of equations (2.37) in block-diagonal form. Doing so results in

\[
\begin{bmatrix}
H & -I & 0 & \cdots & 0 & 0 & 0 \\
-I & H & -I & \cdots & 0 & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & 0 & \cdots & -I & H & -I \\
0 & 0 & 0 & \cdots & 0 & -I & H
\end{bmatrix}
\begin{bmatrix}
T^{n+1}_1 \\
T^{n+1}_2 \\
\vdots \\
T^{n+1}_q \\
p \times N
\end{bmatrix}
= \begin{bmatrix}
T^n_1 & T^n_2 & \cdots & T^n_q \\
B_1 & B_2 & \cdots & B_{N-1} & B_N \\
0 & 0 & \cdots & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & \cdots & 0 & 0
\end{bmatrix}
\begin{bmatrix}
u_1(t) \\
u_2(t) \\
\vdots \\
u_N(t)\\N \times 1
\end{bmatrix}
\]

(2.41)

Using the block-diagonal system (2.41), the solution of the initial value boundary problem is solved by marching in time in increments of \( \Delta t \) and solving the resulting linear system after each time-step. Note that in matrix equation (2.41), the scalar functions \( u_k(t) \) are evaluated
at the $n + 1$ time step. In the work presented here, this is admissible. The functions $u_k(t)$ are prescribed, and their values are known at all times. However, when using this model for control one must make additional adjustments. System (2.41) does not provide a given time value of $T(t, x, y)$ by using only past values of $u_k(s), s \leq t$.

### 2.5 Semi-Discrete Approximation

A semi-discrete approximation is also constructed on the rectangle. Forming difference approximations for the spatial derivatives while leaving the derivative in time undiscretized yields a semi-discrete approximation of the form

$$\frac{\partial}{\partial t} T = AT + BU. \quad (2.42)$$

To construct the $A$ and $B$ matrices in (2.42), the second order spatial derivatives are discretized by difference approximations as

$$\frac{\partial^2}{\partial x^2} T(t, x, y) \approx \frac{T(t, x + \Delta x, y) - 2T(t, x, y) + T(t, x - \Delta x, y)}{\Delta x^2}, \quad (2.43)$$

$$\frac{\partial^2}{\partial y^2} T(t, x, y) \approx \frac{T(t, x, y + \Delta y) - 2T(t, x, y) + T(t, x, y - \Delta y)}{\Delta y^2}. \quad (2.44)$$

Substituting (2.43), (2.44) into the 2-D heat equation results in a semi-discrete approximation at each evaluation point $(x_i, y_j)$ in $\Omega_0$. Implementing uniform spatial step sizes $\Delta x = \Delta y = h$ yields the approximation

$$\frac{\partial}{\partial t} T_{i,j} = \frac{K}{h^2}[-4T_{i,j} + T_{i-1,j} + T_{i+1,j} + (T_{i,j-1}) + (T_{i,j+1})]. \quad (2.45)$$

### 2.5.1 Boundary Conditions

As in the finite difference case, boundary conditions are imposed at specific grid points that correspond to the boundary conditions of the IBVP. As before, if $i = 1$, then $T_{i-1,j} = T_{0,j}$. Similarly, if $i = p$, then $T_{i+1,j} = T_{p+1,j}$. Thus, $i = 1$ or $i = p$ corresponds to grid points
on the lines \( x = a2 \) or \( x = a3 \), respectively. As a result, the temperature at these grid points is held at zero as time evolves. This condition is implemented as

\[
T_{0,j} = 0 \quad \text{and} \quad T_{p+1,j} = 0. \tag{2.46}
\]

A particular value of \( j = 1 \) results in \( T_{i,j-1} = T_{i,0} \). As discussed for the finite difference case, \( T_{i,0} \) corresponds to grid points along \( \Gamma_C \). Thus, the Dirichlet boundary condition imposed by (2.9) is implemented as

\[
T_{i,0} = \sum_{k=1}^{N} b_k(x_i) u_k(t). \tag{2.47}
\]

Finally, if \( j = q \), then \( T_{i,j+1} = T_{i,q+1} \). This corresponds to evaluation points along the line \( y = b2 \). In order for the semi-discrete approximation to agree with the IVBP, it is necessary that

\[
T_{i,q+1} = 0. \tag{2.48}
\]

### 2.5.2 Matrix Equations

Let

\[
D = \begin{bmatrix}
-4 & 1 & 0 & \ldots & \ldots & 0 & 0 & 0 \\
1 & -4 & 1 & \ldots & \ldots & 0 & 0 & 0 \\
0 & 1 & -4 & \ldots & \ldots & 0 & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & 0 & \ldots & \ldots & -4 & 1 & 0 \\
0 & 0 & 0 & \ldots & \ldots & 1 & -4 & 1 \\
0 & 0 & 0 & \ldots & \ldots & 0 & 1 & -4
\end{bmatrix}_{p \times p}
\]

and

\[
T_j = \begin{bmatrix}
T(t, x_1, y_j) \\
T(t, x_2, y_j) \\
T(t, x_3, y_j) \\
\vdots \\
T(t, x_{p-2}, y_j) \\
T(t, x_{p-1}, y_j) \\
T(t, x_p, y_j)
\end{bmatrix}_{p \times 1}
\]

Also, let \( I \) denote the \( p \times p \) identity matrix. Then, allowing \( i \) to range between 1 and \( p \)
for each value of $j$ in (2.45) results in a matrix equation for each $1 \leq j \leq q$ of the form
\[
\frac{\partial}{\partial t} T_j = \frac{K}{h^2} \left[ DT_j + IT_{j-1} + IT_{j+1} \right].
\] (2.50)

In order to write the system of matrix equations (2.50) in block diagonal form, it is necessary to implement the boundary conditions discussed in the previous section. In particular, the time-varying boundary condition along $\Gamma_C$ needs to be incorporated into (2.50). As discussed in Section 2.5.1, grid points along $\Gamma_C$ correspond to $j = 1$. If $j = 1$, then $T_{j-1} = T_0$. Implementing condition (2.47) at each grid point comprising $T_0$ results in a vector of sums of the form
\[
T_0 = \begin{bmatrix}
\sum_{k=1}^{N} b_k(x_1)u_k(t) \\
\sum_{k=1}^{N} b_k(x_2)u_k(t) \\
\sum_{k=1}^{N} b_k(x_3)u_k(t) \\
\vdots \\
\sum_{k=1}^{N} b_k(x_{p-2})u_k(t) \\
\sum_{k=1}^{N} b_k(x_{p-1})u_k(t) \\
\sum_{k=1}^{N} b_k(x_{p})u_k(t)
\end{bmatrix}_{p \times 1}.
\] (2.51)

As in the finite difference case, it is possible to write the vector of sums in (2.51) as the product as a matrix and a vector. This is done in the semi-discrete case exactly the same as it was for the finite difference case. Writing the vector in (2.51) as a matrix product, as in the finite difference case, yields a matrix representation of $T_0$ of the form
\[
T_0 = \begin{bmatrix}
b_1(x_1) & b_2(x_1) & \cdots & b_N(x_1) \\
b_1(x_2) & b_2(x_2) & \cdots & b_N(x_2) \\
b_1(x_3) & b_2(x_3) & \cdots & b_N(x_3) \\
\vdots & \vdots & \ddots & \vdots \\
b_1(x_{p-2}) & b_2(x_{p-2}) & \cdots & b_N(x_{p-2}) \\
b_1(x_{p-1}) & b_2(x_{p-1}) & \cdots & b_N(x_{p-1}) \\
b_1(x_{p}) & b_2(x_{p}) & \cdots & b_N(x_{p})
\end{bmatrix}_{p \times N}
\begin{bmatrix}
u_1(t) \\
u_2(t) \\
u_3(t) \\
\vdots \\
u_{N-2}(t) \\
u_{N-1}(t) \\
u_N(t)
\end{bmatrix}_{N \times 1}.
\] (2.52)
After implementing the matrix representation of $T_0$ given by (2.52), the system of matrix equations (2.50) is written in block diagonal form as

$$\frac{h^2}{K} \begin{bmatrix} \dot{T}_1 \\ T_2 \\ \vdots \\ T_{q-1} \\ T_q \end{bmatrix} = \begin{bmatrix} \mathbf{D} & \mathbf{I} & 0 & \ldots & 0 & 0 & 0 \\ \mathbf{I} & \mathbf{D} & \mathbf{I} & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & \mathbf{I} & \mathbf{D} \\ 0 & 0 & 0 & \ldots & 0 & \mathbf{I} & \mathbf{D} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_{q-1} \\ T_q \end{bmatrix} + \begin{bmatrix} \mathbf{B}_1 & \mathbf{B}_2 & \ldots & \mathbf{B}_{N-1} & \mathbf{B}_N \\ 0 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_{N-1}(t) \\ u_N(t) \end{bmatrix}.$$  

The solution to the initial boundary value problem is computed using the block diagonal system above with Matlab’s ODE45 ordinary differential equation solver. The file ODE45 utilizes a Runge-Kutta algorithm to solve the system of ordinary differential equations above with the specified initial condition $T(0, x, y) = f(x, y)$. 

(2.53)
Chapter 3

PDESolve

The difference approximations on the rectangle produce matrix equations in which the solution to the initial boundary value problem is approximated by solving a linear system at each time-step. The same idea can be applied to rather general domains composed of “rectangular regions”, such as for the geometry of the CVD reactor. A software package called PDESolve®, developed by BEAM Technologies, allows the user to implement such finite difference approximations on more complicated domains. We shall use PDESolve to implement the Crank-Nicholson finite difference approximation on the domain defined by the CVD reactor.

3.1 PDESolve

PDESolve is a tool for rapid development of PDE solvers and simulations. Programs written in PDESolve are much shorter than similar codes written in procedural languages such as FORTRAN. For the most part, the number of lines of code required for a simulation written in PDESolve will be one to two orders of magnitude smaller than FORTRAN code. PDESolve is implemented as a library of C++ classes. Consequently, it offers high-level PDE operations from within a full-featured, object oriented programming language. In other words, PDESolve is able to model mathematical objects, their attributes, behaviors, and the modes of communication between objects.
PDESolve enables the user to describe the partial differential equation and its solution method at a high level. Differential operators and their discretization methods are specified in the PDESolve program. The PDESolve description of the solution is executed as a C++ program.

PDESolve deals with complex geometries, vector and tensor functions and operators, and multiply coupled equations. Results of PDESolve calculations may be visualized with such common packages as TecPlot®, Matlab®, and VTK®. Currently, PDESolve supports finite difference discretization methods, unstructured grid/finite element methods for basis functions up to second order, and boundary elements. Since the work presented here deals with finite difference methods, we focus on an implementation of the Crank-Nicholson finite difference method in PDESolve.

3.2 Creating the Domain

To simulate the initial boundary value problem on a polygonal domain Ω, we construct the domain and define the differential operators that yield the difference approximations. For the CVD geometry, the domain can be constructed as a union of rectangles. This is done by splitting the domain into the five regions illustrated in Figure 3.1.

Region 1 = \([a_1, a_2] \times [b_1, b_2]\), \quad Region 2 = \([a_1, a_2] \times [b_2, b_3]\),

Region 3 = \([a_2, a_3] \times [b_2, b_3]\), \quad Region 4 = \([a_3, a_4] \times [b_2, b_3]\),

Region 5 = \([a_3, a_4] \times [b_1, b_2]\).

The domain Ω is written as the union of five regions of the form

Ω = Region1 ∪ Region2 ∪ Region 3 ∪ Region 4 ∪ Region 5.

This is analogous to the way Ω is constructed with PDESolve. In PDESolve, Ω is also constructed as the union of these five regions. It is done with the SpatialDomain command
by forming each region as a Cartesian product. For example, Region 1 in Figure 3.1 is created by the command \texttt{SpatialDomain(a1,a2)*SpatialDomain(b1,b2)}. Taking the sum of the five regions results in the entire computational domain. This is illustrated in the following line of code:

\begin{verbatim}
SpatialDomain dom = SpatialDomain(a1,a2)*SpatialDomain(b1,b2) +
                  SpatialDomain(a1,a2)*SpatialDomain(b2,b3) +
                  SpatialDomain(a2,a3)*SpatialDomain(b2,b3) +
                  SpatialDomain(a3,a4)*SpatialDomain(b2,b3) +
                  SpatialDomain(a3,a4)*SpatialDomain(b1,b2).
\end{verbatim}
The code above constructs the computational domain associated with the geometry of the CVD reactor and assigns it to the variable dom.

### 3.3 Partitioning the Domain

Once the domain is created, it must be partitioned in order to implement the finite difference approximation. This is done by specifying the finite difference mesh on each of the five regions individually. This is done in PDESolve with the `FDMesh` and `FDMeshSpec` commands. `FDMeshSpec` assigns the number of partitions on each block in both the $x$ and $y$ directions. With $m$ and $n$ denoting the number of evaluation points in the $x$ and $y$ directions respectively, \( FDMeshSpec(m_1)*FDMeshSpec(n_1) \) partitions Region 1 in both the $x$ and $y$ directions. Here, $m_1$ is the number of evaluation points in the $x$ direction and $n_1$ is the number of evaluation points in the $y$ direction. The meshes on the other blocks are created in a completely analogous way. The union of the individual partitioned blocks then gives the finite difference mesh `FDMesh`.

The following line of code illustrates how to create the finite difference mesh on the computational domain with the geometry of the reactor. The mesh on this geometry is specified as

\[
\text{FDMesh probgrid(dom, FDMeshSpec}(m_1)\text{*FDMeshSpec}(n_1) + \\
\text{FDMeshSpec}(m_2)\text{*FDMeshSpec}(n_2) + \\
\text{FDMeshSpec}(m_3)\text{*FDMeshSpec}(n_3) + \\
\text{FDMeshSpec}(m_4)\text{*FDMeshSpec}(n_4) + \\
\text{FDMeshSpec}(m_5)\text{*FDMeshSpec}(n_5)).
\]

The code above defines a finite difference mesh on the computational domain `dom`. The name of the mesh given by the code is `probgrid`. It should be noted that the order in which
the mesh is created is important. If the computational domain is created by forming Region 1 and then Region 2, Region 3, etc., then the finite difference mesh has to be constructed on Region 1 and then Region 2, Region 3, etc.

3.4 Spatial Derivatives

After constructing the computational domain, approximations to the second order spatial derivatives are constructed. This is done very simply in PDESolve. The \texttt{DiffOp} command lets the user specify the order of the derivative, the spatial direction of the derivative, and the truncation error needed in the difference approximation to the derivative. The parameters (order of derivative, spatial direction, order of truncation error) are used when a \texttt{DiOp} is constructed. Simply specifying \texttt{DiffOp dxx(2,0,2)} yields the second order difference operator \texttt{dxx}, corresponding to finite difference approximation (2.12), in the \texttt{x} direction with a second order truncation error. Note that the \texttt{x} direction is specified by a zero, as indices begin with zero in C++. \texttt{DiffOp dyy(2,1,2)} yields the second order difference operator \texttt{dyy}, corresponding to difference equation (2.13), in the \texttt{y} direction with a second order truncation error. Setting $T_{n+1} = T_{\text{next}}$, $T_n = T_{\text{current}}$, and letting $dt$ denote the time step $\Delta t$, the Crank-Nicholson scheme can be written in PDESolve as

\[
\text{Expr pdo} = T_{\text{next}} - T_{\text{current}} - \frac{K \cdot dt}{2} \cdot \left( dxx \cdot T_{\text{next}} + dxx \cdot T_{\text{current}} \right) \\
- \frac{K \cdot dt}{2} \cdot \left( dyy \cdot T_{\text{next}} + dyy \cdot T_{\text{current}} \right).
\]

Above, pdo is the name given to the difference approximation in the PDESolve code. Note that the discretization method is written in a homogeneous fashion when specified in PDESolve.
3.5 Boundary Conditions

Now that the finite difference approximation and the computational domain have been constructed in PDESolve, the only remaining aspects of the initial value boundary problem to be constructed are the boundary conditions. This is done in PDESolve with the BC command.

In PDESolve, boundary conditions are formed in the same fashion as the construction of the computational domain, namely by writing the boundary conditions in the form of a Cartesian product. For example, if a computational domain of \([0,1] \times [0,1]\) is constructed, and \(u\) is the name given to the solution of a partial differential equation on this domain, then

\[
BC \text{ SIDES} = BC(u==0, u==0)\times BC(u==0,u==0)
\]

specifies that the solution remain fixed at zero along the boundary. In this example, SIDES is the name assigned to the boundary condition. If the computational domain can be constructed by splitting it into several regions, the boundary condition on the entire domain is written as the sum of the boundary conditions on each of the regions.

To impose boundary conditions on the somewhat complicated domain representing the geometry of the reactor, boundary conditions are specified on the boundary of each of the five blocks of the partitioned domain individually. Two blocks that share a common boundary also share a common boundary condition. In PDESolve, boundary conditions on blocks that share a common “internal boundary” are handled with the LinkAcross command.

LinkAcross links together unknown functions and their derivatives across “internal boundaries” in a multiblock structure. In order for LinkAcross to be used, integer labels are given to shared boundaries. These labels enable LinkAcross to specify matching boundary conditions on blocks that share a common boundary. The integer label and the name assigned to the numerical solution of the partial differential equation are the arguments given to LinkAcross when it is called.

As an example, suppose a finite difference grid has been constructed in PDESolve on
the domain $([0,1] \times [0,1]) \cup ([1,2] \times [0,1])$. Then, by assigning the common boundary of the computational domain an integer label of 1, boundary conditions are specified on each of the blocks $[0,1] \times [0,1]$ and $[1,2] \times [0,1]$ individually. If the numerical solution on this computational domain is named HEAT, and the exterior boundary condition is that the temperature of the exterior boundary remain fixed at zero as time evolves, then the commands

\[
\text{BC block1} = \text{BC}(\text{HEAT}==0, \text{LinkAcross}(1,\text{HEAT})) \ast \text{BC}(\text{HEAT}==0, \text{HEAT}==0)
\]

\[
\text{BC block2} = \text{BC}(\text{LinkAcross}(1,\text{HEAT}), \text{HEAT}==0) \ast \text{BC}(\text{HEAT}==0, \text{HEAT}==0)
\]

specify the boundary conditions on all four sides of each individual block. The boundary conditions on the region $[0,1] \times [0,1]$ are assigned to the variable block1. Similarly, the boundary conditions on the region $[1,2] \times [0,1]$ are assigned to the variable block2. After the boundary conditions are specified on each of the blocks individually, boundary conditions on the entire computational domain are written as the sum of the boundary conditions of each individual block. Thus, the command

\[
\text{BC total\_boundary\_condition} = \text{block1} + \text{block2}
\]

creates the desired boundary condition on the entire computational domain. As can be seen, fairly complex geometries are handled easily by the LinkAcross command.

The construction of the boundary conditions on the geometry representing the reactor is done in a similar way as discussed in the example above. Integer labels are given to boundaries that are shared between two regions comprising the computational domain.

Utilizing the integer tags illustrated by the arrows in Figure 3.2, boundary conditions (2.6), (2.7) in the initial boundary value problem are specified in a similar fashion as in the example discussed above.

Let bc1, bc2, bc3, bc4, and bc5 represent the boundary conditions on Region 1, Region 2, Region 3, Region 4, and Region 5, respectively. Then, bc3 must incorporate the time-varying boundary condition representing the influence of the heating lamps on $\Gamma_C$. Writing the temperature profile function $b_k(x)$ as lampk and the scalar function $u_k(t + \Delta t)$ as uk
allows for the implementation of the time-varying boundary condition on $\Gamma_C$. That is, $T_{\text{next}} = \sum_{k=1}^{N} \text{lamp}_k(u_k)$ along $\Gamma_C$, where $N$ is the number of heating lamps.

With the boundary condition along $\Gamma_C$ constructed in this way, the PDEsolve assignment statements needed to specify the boundary conditions on each of the individual regions of the domain for four lamps ($N=4$) are written as

$$BC \ bc1 = BC(T_{\text{next}} == 0, T_{\text{next}} == 0) \ast BC(T_{\text{next}} == 0, \text{LinkAcross}(1, T_{\text{next}})),$$

$$BC \ bc2 = BC(T_{\text{next}} == 0, \text{LinkAcross}(2, T_{\text{next}})) \ast BC(\text{LinkAcross}(1, T_{\text{next}}), T_{\text{next}} == 0),$$

$$BC \ bc3 = BC(\text{LinkAcross}(2, T_{\text{next}}), \text{LinkAcross}(3, T_{\text{next}}))$$

$$\ast BC(T_{\text{next}} == (u1*\text{lamp1} + u2*\text{lamp2} + u3*\text{lamp3} + u4*\text{lamp4}), T_{\text{next}} == 0),$$
BC \text{bc}4 = BC(\text{LinkAcross}(3, T_{\text{next}}), T_{\text{next}}==0) \times BC(\text{LinkAcross}(4, T_{\text{next}}), T_{\text{next}}==0),

BC \text{bc}5 = BC(T_{\text{next}}==0, T_{\text{next}}==0) \times BC(T_{\text{next}}==0, \text{LinkAcross}(4, T_{\text{next}})).

With the boundary conditions specified on each block, the boundary conditions for the entire domain are written as the sum of the boundary conditions on all of the blocks as

$$BC \text{bc} = \text{bc}1 + \text{bc}2 + \text{bc}3 + \text{bc}4 + \text{bc}5.$$ 

Above, \text{bc} is the name assigned to the boundary condition over the entire domain.

### 3.6 Solving the System

Once the finite difference mesh, discretization of the partial differential equation, boundary conditions, and the name assigned to the computed solution have been specified, the resulting linear system is solved in PDESolve with the \texttt{LinearProblem} command. The command \texttt{LinearProblem prob(probgrid,pdo,bc,T_{\text{next}})} assigns the linear problem resulting from the discretization to the variable \texttt{prob}. Simply specifying \texttt{T_{\text{next}}=prob.solve()} solves the linear system and assigns the result to \texttt{T_{\text{next}}}.
Chapter 4

Numerical Experiments

4.1 Numerical Tests

Now that numerical schemes have been developed to approximate the solution to the initial boundary value problem, it is important to test these schemes for convergence. To do this, an exact solution is chosen and substituted into the homogeneous 2-D heat equation. This yields a forcing function which is set as the right hand side of the homogeneous equation. For the numerical tests presented here, we specify the exact solution as

\[ T(t, x, y) = e^{-Kt} \sin(x) \cos(y). \]  

Substituting the exact solution into the 2-D heat equation requires derivative calculations in time and space. Taking the first derivative in time of \( T(t, x, y) \) results in

\[ \frac{\partial}{\partial t} T(t, x, y) = -KT(t, x, y). \]  

Similarly, second order derivatives are taken in the x and y directions. These calculations yield

\[ \frac{\partial^2}{\partial x^2} T(t, x, y) = -T(t, x, y), \]  

\[ \frac{\partial^2}{\partial y^2} T(t, x, y) = -T(t, x, y). \]
Substituting (4.2)-(4.4) into the homogeneous 2-D heat equation yields a nonhomogeneous equation with a forcing function in terms of the exact solution $T(t, x, y)$. The equation resulting from this substitution is

$$\frac{\partial}{\partial t}T(t, x, y) - K \left( \frac{\partial^2}{\partial x^2}T(t, x, y) + \frac{\partial^2}{\partial y^2}T(t, x, y) \right) = KT(t, x, y). \quad (4.5)$$

In order to utilize (4.5) to obtain a forcing function for the semi-implicit Crank-Nicholson scheme, it is necessary to form the average of $KT(t, x, y)$ at time-steps $n$ and $n+1$. Doing so results in the forcing function $g(t, x, y)$ of the form

$$g(t, x, y) = \frac{K}{2} (T(t, x, y) + T(t + \Delta t, x, y)). \quad (4.6)$$

Thus, the forcing value on each evaluation point $(x_i, y_j)$ on the grid for the Crank-Nicholson scheme is $g(t, x_i, y_j)$.

The final step in comparing the exact solution to the numerical solutions is the implementation of the correct initial condition in the approximation schemes. At the value $t=0$, an initial condition is obtained from the exact solution. Specifying $t = 0$ in (4.1) results in

$$T(0, x, y) = \sin(x) \cos(y). \quad (4.7)$$

Therefore, an initial condition of $f(x, y) = \sin(x) \cos(y)$ is used in the numerical schemes.

Tests are performed on the finite difference approximation on the rectangle as well as the semi-discrete approximation on the same domain. A comparison of results obtained from PDESolve is also compared to the exact solution on the more complex polygonal domain. In all tests, the same exact solution $T(t, x, y) = e^{-Kt} \sin(x) \cos(y)$ with $K = 0.3$ is used. In addition, the geometry of the computational domains will be chosen in such a way as to yield $\sin(x) \cos(y) = 0, \forall (x, y) \in \partial\Omega$.

### 4.1.1 Numerical Tests on the Square

Comparisons are done between the finite difference scheme and the semi-discrete approximation with the exact solution on the square $([0, \pi]) \times ([0, \pi])$. The numerical solutions
are presented at $t=1$. The time-step sizes used are $\Delta t=0.2$, 0.1, and .05. The spatial-step sizes used are $\Delta x = \Delta y = 0.1$. The semi-discrete approximation is independent of $\Delta t$, as illustrated in (2.53). Thus, reducing the time-step $\Delta t$ in the Crank-Nicholson scheme will provide a comparison of accuracy between the semi-discrete and the Crank-Nicholson schemes. Figures 4.4, 4.5, 4.6 are comparisons with $\Delta t = 0.2$, 0.1, and 0.05, respectively.
Figure 4.1: Exact and Computed Solutions on the Square with $\Delta t = 0.2$
The exact solution $T(t,x,y) = e^{-kt} \cos(y) \sin(x)$ at $t = 1$.

Finite Difference Approximation to $T(t,x,y) = e^{-kt} \cos(y) \sin(x)$ at $t = 1$, $\Delta t = 0.1$.

Semi-Discrete Approximation to $T(t,x,y) = e^{-kt} \cos(y) \sin(x)$ at $t = 1$.

Figure 4.2: Exact and Computed Solutions on the Square with $\Delta t = 0.1$
Figure 4.3: Exact and Computed Solutions on the Square with $\Delta t = 0.05$
From the numerical tests on the square, the maximum absolute value of the difference between the finite difference method (F.D.) and the exact solution is calculated. The same is also done for the semi-discrete method (S.D.). The values obtained are in the table below.

| $\Delta t$ / $\max|F.D. - Exact|$ on $\Omega_0$ | $\max|S.D. - Exact|$ on $\Omega_0$ |
|-----------------|-----------------|
| 0.2 / 1.24e-02 | 2.19e-02 |
| 0.1 / 6.80e-03 | - |
| 0.05 / 4.0e-03 | - |

From the table, as the time-step size decreased, the finite difference solution became a better approximation of the true solution. All results obtained from the finite difference scheme yielded more accurate solutions than those obtained from the semi-discrete scheme.

### 4.1.2 Numerical Tests on the Entire Domain

A numerical test is conducted on the PDESolve implementation of the Crank-Nicholson method on the more complicated domain representing the reactor. The computational domain used is

$\left([[-\pi, 0] \times \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]) \cup ([[-\pi, 0] \times \left[\frac{\pi}{2}, \frac{3\pi}{2}\right]) \cup ([0, \pi] \times \left[\frac{\pi}{2}, \frac{3\pi}{2}\right]) \cup ([\pi, 2\pi] \times \left[\frac{\pi}{2}, \frac{3\pi}{2}\right])\cup ([\pi, 2\pi] \times [-\frac{\pi}{2}, \frac{\pi}{2}]).$ \]

This corresponds to values of $a_1=-\pi$, $a_2=0$, $a_3=\pi$, $a_4=2\pi$, $b_1=-\frac{\pi}{2}$, $b_2=\frac{\pi}{2}$, $b_3=\frac{3\pi}{2}$.

Plots of the exact solution $T(t, x, y)$, the computed solution $T_{\text{approx}}(t, x, y)$, and the difference $E(t, x, y) = T(t, x, y) - T_{\text{approx}}(t, x, y)$ are presented at $t=1$. The time-step sizes used are $\Delta t=0.2$, 0.1, and .05. The spatial-step sizes used are $\Delta x = \Delta y = 0.2$. Figures 4.4, 4.5, and 4.6 are comparisons of the computed and exact solutions with $\Delta t=0.2$, 0.1, and .05, respectively.
The exact solution \( T(t,x,y) = e^{-Kt} \sin(x) \cos(y), K=0.3, t=1 \)

The numerical approximation \( T_{\text{approx}}(t,x,y) \) to the exact solution, \( K=0.3, \Delta t = 0.2, t=1 \)

\[ E(t,x,y) = T(t,x,y) - T_{\text{approx}}(t,x,y) \]

Figure 4.4: Exact and Computed Solution on the Entire Domain with \( \Delta t = 0.2 \)
The exact solution $T(t,x,y) = e^{-Kt} \sin(x) \cos(y)$, $K=0.3$, $t=1$

The numerical approximation $T_{\text{approx}}(t,x,y)$ to the exact solution, $K=0.3$, $\Delta t = 0.1$, $t=1$

$E(t,x,y) = T(t,x,y) - T_{\text{approx}}(t,x,y)$

Figure 4.5: Exact and Computed Solution on the Entire Domain with $\Delta t = 0.1$
The exact solution $T(t,x,y) = e^{-Kt} \sin(x) \cos(y)$, $K=0.3$, $t=1$

The numerical approximation $T_{\text{approx}}(t,x,y)$ to the exact solution, $K=0.3$, $\Delta t = 0.05$, $t=1$

$E(t,x,y) = T(t,x,y) - T_{\text{approx}}(t,x,y)$

Figure 4.6: Exact and Computed Solution on the Entire Domain with $\Delta t = 0.05$
From the results of the numerical test, the maximum absolute value of the difference between the exact solution and the approximated solution is calculated. These values are in the table below.

Table 4.2: Errors on the Entire Domain

| $\Delta t$ | $\max |T(t, x, y) - T_{approx}(t, x, y)|$ on $\Omega$ |
|------------|---------------------------------|
| 0.2        | 7.90e-03                        |
| 0.1        | 2.40e-03                        |
| 0.05       | 4.43e-04                        |

From the table, as the time-step size decreases, the value of the error decreases dramatically. Convergence of the numerical solution to the exact solution is apparent.
4.2 Geometry Experiments

To investigate the effect of geometry on thermal conduction throughout the reactor, numerical experiments are performed. The entire boundary $\partial \Omega$ is held at zero temperature as time evolves. No heat lamps are implemented along $\Gamma_C$. The values $a_1$, $a_2$, $a_3$, $a_4$, $b_1$, $b_2$, and $b_3$ defining the computational domain are varied to illustrate the importance of domain geometry. Snapshots of the solution at times 2, 4, 8 are presented. The initial condition used is $T(0, x, y) \equiv 1$. In addition, calculations are iterated until the temperature throughout the domain satisfies $T(t, x, y) \leq 10^{-6}$. The solution at this value of time is presented. The step-size values used in the experiments are $\Delta t = \Delta x = \Delta y = 0.2$, and $K = 0.5$.

The values of $a_1$, $a_2$, $a_3$, $a_4$, $b_1$, $b_2$, and $b_3$ used in the geometry experiments are represented in the following table.

<table>
<thead>
<tr>
<th>Geometry Experiment</th>
<th>a1</th>
<th>a2</th>
<th>a3</th>
<th>a4</th>
<th>b1</th>
<th>b2</th>
<th>b3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-5</td>
<td>-3</td>
<td>3</td>
<td>5</td>
<td>-8</td>
<td>-1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>-5</td>
<td>-3</td>
<td>3</td>
<td>7</td>
<td>-8</td>
<td>-1</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4.3: Values Used in Geometry Experiments
Figure 4.7: Geometry Experiment 1 at t=0. (Initial Condition)

Figure 4.8: Geometry Experiment 1 at t=2.
Figure 4.9: Geometry Experiment 1 at t=4.

Figure 4.10: Geometry Experiment 1 at t=8.
Figure 4.11: Geometry Experiment 1 at $t=40.6$. 
Figure 4.12: Geometry Experiment 2 at t=0. (Initial Condition)

Figure 4.13: Geometry Experiment 2 at t=2.
Figure 4.14: Geometry Experiment 2 at t=4.

Figure 4.15: Geometry Experiment 2 at t=8.
Figure 4.16: Geometry Experiment 2 at t=50.2.
4.3 Numerical Experiments with the Lamps

The influence of the heating lamps on the conduction of thermal energy throughout the reactor can be seen by implementing the time-varying boundary condition along $\Gamma_C$. After multiplying the temperature profile functions by the scalar functions $u_k(t)$ shown below, the product $b_k(x)u_k(t)$ models the k-th lamp being “off” until the time $t_k$. At $t_k$, the k-th lamp turns on and increases the temperature along $\Gamma_C$ in a linear fashion in time until the lamp reaches its maximum temperature.

![Figure 4.17: A Sample $u_k(t)$](image)

Once the maximum temperature of the lamp has been attained, the temperature of the lamp remains at the maximum value for the remaining duration of the experiment. In the numerical experiments presented here, the maximum temperature of each lamp is unity, and the linear portion of each $u_k(t)$ has slope one. Thus, the amount of time required for a lamp to attain maximum heat from the time it is turned on is one time unit.

For the numerics, the values used are $\Delta x = \Delta y = 0.2$, $\Delta t = 0.2$, and $K = 0.5$. The terminating time of each experiment is eight seconds. Snapshots of the solution at times 2, 4, 8 are presented, along with a plot of the initial condition. The initial condition used is $T(0, x, y) \equiv 1$. The geometries of Geometry Experiments 1 and 2 are used. Numerical
experiments are conducted on these geometries for one and four lamps along $\Gamma_C$. The time value $t_k$ at which the k-th lamp “turns on” and the geometry used in each numerical experiment are represented in the following table.

Table 4.4: Values Used in Numerical Experiments

<table>
<thead>
<tr>
<th>Numerical Experiment</th>
<th>Number of Lamps</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>Geometry Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>geometry 1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>geometry 1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>geometry 2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>geometry 2</td>
</tr>
</tbody>
</table>
Figure 4.18: Numerical Experiment 1 at t=0. (Initial Condition)

Figure 4.19: Numerical Experiment 1 at t=2.
Figure 4.20: Numerical Experiment 1 at $t=4$.

Figure 4.21: Numerical Experiment 1 at $t=8$. 
Figure 4.22: Numerical Experiment 2 at t=0. (Initial Condition)

Figure 4.23: Numerical Experiment 2 at t=2.
Figure 4.24: Numerical Experiment 2 at \( t=4 \).

Figure 4.25: Numerical Experiment 2 at \( t=8 \).
Figure 4.26: Numerical Experiment 3 at t=0. (Initial Condition)

Figure 4.27: Numerical Experiment 3 at t=2.
Figure 4.28: Numerical Experiment 3 at $t=4$.

Figure 4.29: Numerical Experiment 3 at $t=8$. 
Figure 4.30: Numerical Experiment 4 at t=0. (Initial Condition)

Figure 4.31: Numerical Experiment 4 at t=2.
Figure 4.32: Numerical Experiment 4 at t=4.

Figure 4.33: Numerical Experiment 4 at t=8.
Chapter 5

Conclusions

5.1 Overview of Results

In this paper, we developed a finite difference and a semi-discrete method to approximate the solution of the initial boundary value problem

\[
\frac{\partial}{\partial t} T(t, x, y) = K \left( \frac{\partial^2}{\partial x^2} T(t, x, y) + \frac{\partial^2}{\partial y^2} T(t, x, y) \right), \quad t > 0, \quad (x, y) \in \bar{\Omega}, \quad (5.1)
\]

\[
T(0, x, y) = f(x, y), \quad (5.2)
\]

\[
T(t, x, y) \bigg|_{\Gamma_C} = \sum_{k=1}^{N} b_k(x) u_k(t), \quad t > 0, \quad (5.3)
\]

\[
T(t, x, y) \bigg|_{\Gamma_U} = 0, \quad t > 0. \quad (5.4)
\]

In (5.2), \(f(x, y)\) is the initial temperature distribution throughout the domain. The system above models thermal conduction in a chemical vapor deposition reactor in which a heated substrate is exposed to a gas containing precursor molecules. The result is the deposition of a thin film of a desired compound on the substrate.

The methods were implemented in PDESolve\textsuperscript{©}, a high level software package. The goal of this work was two-fold: (i) to provide a reasonable “test bed” problem for evaluating the
usefulness of PDESolve and (ii) to study the impact of geometry on thermal conduction. The second objective is intended as a first step in a long term effort to design and control actual CVD reactors.

Matrix equations resulting from the two approximation methods were developed on the rectangle \([a_2,a_3] \times [b_2,b_3]\). Numerical tests were performed with the two methods on the rectangle to illustrate accuracy of approximations. The semi-discrete approximation was fairly accurate, but had the disadvantage that greater accuracy could not be achieved. There is no time-step \(\Delta t\) involved in the semi-discrete approximation. This is a result of the lack of a difference approximation of the derivative in time. The finite difference method implemented, commonly known as a Crank-Nicholson method, was also accurate and had the advantage that refinements in the time-step led to greater accuracy of the computed solution.

After numerical accuracy of the approximating schemes was demonstrated, geometry experiments were performed on the more complex computational domain representing the geometry of the reactor. The geometry experiments illustrate the significance of domain geometry with regard to thermal conduction throughout the domain. Different geometries led to a variety of numerical solutions. This information provides insight into the best design geometry of the reactor.

Numerical experiments were also performed on the more complex region to illustrate the influence of the time-varying boundary condition of the conduction of thermal energy. Implementing different numbers of lamps on different geometries also provided a variety of results. Utilizing this information in addition to the results of the geometry experiments provides a better understanding of the best way of keeping the temperature of a portion of the domain at a prescribed temperature. These results will eventually aid in the development of optimal control and design tools to be used on a more realistic reactor model.
5.2 Conclusions

The numerical results contained in this paper illustrate the importance of domain geometry, boundary conditions, and initial condition on the two dimensional heat equation. Different domain geometries led to computed solutions with different characteristics. In addition, implementing different boundary conditions led to computed solutions with different temperature distributions throughout the domain. Obviously, the geometry of the domain and the boundary conditions imposed on it are important. PDESolve proved to be an extremely useful tool for the simulations. However, the control problem will require more effort and additional modifications to PDESolve software.

5.3 Future Work

While the results presented in this paper are interesting and useful in their own right, they are but a first step in the long term goal of developing optimal design and control tools for a physical reactor. There is still work to be done in achieving this goal. For one, the temperature profile functions used to approximate the influence of the heating lamps on the boundary $\Gamma_C$ are somewhat simplistic. The physics of radiation heat transfer must be incorporated into the model of the lamps.

In the reactor, gas flows over a heated substrate. In the model presented in this paper, thermal conduction is the only aspect being investigated. In the physical system, however, the heating and cooling of the gas causes the gas to flow from one place to another inside the reactor. Thus, the model described in this paper is not yet complete. The two dimensional heat equation is not sufficient to describe all of the processes taking place inside the reactor. Equations describing the flow of the gas need to be included in the model in order for it to better describe the processes occurring inside the reactor.

Optimal design of the reactor is also an important goal. Investigating how sensitive the temperature distribution is to small changes in domain geometry will provide information as
to the best shape of the reactor. It is important that small perturbations in the parameters
defining the domain do not lead to large differences in the conduction of thermal energy
throughout the reactor. As a result, sensitivity analysis will be useful in finding the best
geometry for the reactor.
Bibliography


Appendix A

PDE Solve Source Codes

A.1 PDE Solve implementation of the Crank-Nicholson scheme on the polygonal domain

// Include C++ header files
#include "PDESolve.h"
#include <math.h>
#include <iomanip.h>
#include <iostream.h>

int count = 1;
const Real K=.2; // Diffusivity Constant
const Real dt=.2; // Time step
const Real tfinal=10.01; // Terminating time

Real heatmax1=1; // The maximum heat value of lamp1.
Real heatmax2=1; // The maximum heat value of lamp2.
Real heatmax3=1; // The maximum heat value of lamp3.
Real heatmax4=1; // The maximum heat value of lamp4.

const int N=4; // The number of lamps. There can be at most 4 lamps.

Real tswitch1a = 0; // The time value the first lamp turns on.
Real tswitch2a = 2; // The time value the second lamp turns on.
Real tswitch3a = 4; // The time value the third lamp turns on.
Real tswitch4a = 6; // The time value the fourth lamp turns on.

Real u1 = 0;
Real u2 = 0; // The controllers for each of the lamps. The initial value
Real u3 = 0; // for each control is 0, since the earliest they can be
Real u4 = 0; // turned on is at t=0.

const Real a1=-5;
const Real a2=-3;
const Real a3=3;
const Real a4=7; // Constants defining the polygonal domain
const Real b1=-8;
const Real b2=-1;
const Real b3=3;

int chopy=6; // The number of evaluation points in y per unit length
int chopx=6; // The number of evaluation points in x per unit length

int chop1x=floor(a2-a1)*chopx;
int chop2x=floor(a3-a2)*chopx; // Determine the number of evaluation points
int chop3x=floor(a4-a3)*chopx; // in each part of the domain.
int chop1y=floor(b2-b1)*chopy;
int chop2y=floor(b3-b2)*chopy;

// Boundary and forcing functions

Real frhs(const Coords& x) {
    return 0;
}

Real lamp_one(const Coords& x) {
    if ((x[0] >= a2) && (x[0] <= (a2+(a3-a2)/N))
        return heatmax1*fabs(sin(((N*M_PI)/(a3-a2))*(x[0]-a2)));
    else return 0;
}

Real lamp_two(const Coords& x) {
    if ((x[0] >= (a2+(a3-a2)/N)) && (x[0] <=(a2+2*(a3-a2)/N))
        return heatmax2*fabs(sin(((N*M_PI)/(a3-a2))*(x[0]-a2)));
    else return 0;
}

Real lamp_three(const Coords& x) {
    if ((x[0] >= (a2+2*(a3-a2)/N)) && (x[0] <= (a2+3*(a3-a2)/N))
        return heatmax3*fabs(sin(((N*M_PI)/(a3-a2))*(x[0]-a2)));
    else return 0;
}
Real lamp_four(const Coords& x) {
    if((x[0] >= (a2+3*(a3-a2)/N)) && (x[0] <= a3))
        return heatmax4*fabs(sin(((N*M_PI)/(a3-a2))*(x[0]-a2)));
    else return 0;
}

Real r1(const Coords& x) {
    return 0;
}

// Initial Condition

Real T0(const Coords& x) {
    return 1;
}

int main(void)
{
    // Establish the polygonal domain
    SpatialDomain dom =
        SpatialDomain(a1,a2)*SpatialDomain(b1,b2) +
        SpatialDomain(a1,a2)*SpatialDomain(b2,b3) +
        SpatialDomain(a2,a3)*SpatialDomain(b2,b3) +
        SpatialDomain(a3,a4)*SpatialDomain(b2,b3) +
        SpatialDomain(a3,a4)*SpatialDomain(b1,b2);

    // Discretize the domain

    FDMesh probgrid(dom,FDMeshSpec(chop1x)*FDmeshSpec(chop1y)+
        FDMeshSpec(chop1x)*FDmeshSpec(chop2y)+
        FDMeshSpec(chop2x)*FDmeshSpec(chop2y)+
        FDMeshSpec(chop3x)*FDmeshSpec(chop2y)+
        FDMeshSpec(chop3x)*FDmeshSpec(chop1y));

    Function lamp1(1,Scalar,lamp_one);
    Function lamp2(1,Scalar,lamp_two);
    Function lamp3(1,Scalar,lamp_three); // Evaluate defining functions
    Function lamp4(1,Scalar,lamp_four);
    Function B1(2,Scalar,r1);
    Function f(2,Scalar,frhs);
    Function T_current(2,Scalar,T0);
    Function T_next(2,Scalar);
Function initial(2,Scalar,T0);

// Define the differential operators in terms of
// (order, direction, order of truncation error),

DiffOp dxx(2,0,2);
DiffOp dyy(2,1,2);

ofstream controlFile("timeStepHeat"); // File to store movie frames.
FDDiscretize fd(probgrid);
Function init=fd.discretize(initial);
ofstream firstFrame("timeStepHeat.frame0"); // Write the initial condition
firstFrame<<init<<endl; // to a file.
controlFile<<"timeStepHeat.frame0"<<endl;

for (Real time=dt; time<=tfinal; time=time+dt)
{

// MOVIE STORAGE

char filename[100];
sprintf(filename,"timeStepHeat.frame%d",count);
PDEString fname(filename);

// Decide what the value of each control is.

if (time < tswitch1a) u1=0;
   if(time >= tswitch1a) u1=(time-tswitch1a);
      if((time-tswitch1a) >= heatmax1) u1=1;

if (time < tswitch2a) u2 = 0;
   if(time >= tswitch2a) u2=(time-tswitch2a);
      if((time-tswitch2a) >= heatmax2) u2=1;
      if(N<2) u2=0;

if (time < tswitch3a) u3=0;
   if(time >= tswitch3a) u3=(time-tswitch3a);
      if ((time-tswitch3a) >= heatmax3) u3 = 1;
      if(N<3) u3=0;

if (time < tswitch4a) u4=0;
   if(time >= tswitch4a) u4 = (time-tswitch4a);
      if ((time - tswitch4a) >= heatmax4) u4 = 1;
      if(N<4) u4=0;

//
cout<<"    "<<endl;
cout<<"    "<<endl;
cout<<"Time: "<<time<<endl;
cout<<"Control_1: "<<u1<<endl;
cout<<"Control_2: "<<u2<<endl;
cout<<"Control_3: "<<u3<<endl;
cout<<"Control_4: "<<u4<<endl;

Expr pdo =
    T_next-T_current-(K*dt/2)*(dxx*T_next+dxx*T_current)
    -(K*dt/2)*(dyy*T_next+dyy*T_current)-f;

    // Establish the boundary conditions
    BC bc1=BC(T_next==B1,T_next==B1)*BC(T_next==B1,LinkAcross(1,T_next));
    BC bc2=BC(T_next==B1,LinkAcross(2,T_next))  
        *BC(1,T_next),T_next==B1);
    BC bc3=BC(LinkAcross(2,T_next),LinkAcross(3,T_next))  
        *BC(T_next==(u1*lamp1 + u2*lamp2 + u3*lamp3 +u4*lamp4),T_next==B1);
    BC bc4=BC(LinkAcross(3,T_next),T_next==B1)  
        *BC(4,T_next),T_next==B1);
    BC bc5=BC(T_next==B1,T_next==B1)*BC(T_next==B1,LinkAcross(4,T_next));

    BC bc=bc1+bc2+bc3+bc4+bc5;

    LinearProblem prob(probgrid,pdo,bc,T_next);  // Solve the problem
    T_current=prob.solve();

    // MORE MOVIE STORAGE

    ofstream frameFile(filename);
    frameFile<<T_current<<endl;
    controlFile<<fname<<endl;

    //
    count = count+1;

}
Appendix B

Matlab Source Codes

B.1 Codes to construct and solve the semi-discrete approximation on the square

clear all
global A B N u n x y K;
a1 = 0;
a2 = pi;
b1 = -pi/2;
b2 = pi/2;
N = 3;
h = .1;
K=.3;
dt=.05;
tfinal = 1;
x=[a1+h:h:a2-h]’;
y=[b1+h:h:b2-h]’;
m = length(x);
n = length(y);

D=toeplitz([-4 1 zeros(1,(m-2))]);
I = eye(m);
A = [D I zeros(m,(n*m - 2*m)) ; I D I zeros(m,(n*m - 3*m))];

for k = 1:1:n-3;
A = [A;zeros(m,k*m) I D I zeros(m,n*m - k*m - 3*m)];
end;
A=(K/(h^2))*[A;zeros(m,n*m-2*m) I D];
for k = 1:length(x);
[l1 l2 l3 l4] = bi(N,a1,a2,x(k,1));
B0(k,:) = [l1 l2 l3 l4];
end;

for i=1:N;
B(:,i) = B0(:,i);
end;

B=(K/(h^2))*[B;zeros(m*(n-1),N)];
tspan=[0 :dt : tfinal];
w0=exact(x,y);
[t,w] = ode45('rhs',tspan,w0);
sol=w(length(tspan),:);

for k=0:1:n-1;
Sol(k+1,:) = sol([k*m+1:m*(k+1)]);
end;

for d=1:N;
profile(:,d) = B0(:,d);
end;

Sol=[[profile*u’;Sol;zeros(1,m)];
Sol=[zeros(n+2,1) Sol zeros(n+2,1)]

figure(1);
clear;

subplot(3,1,2);
PDEview('heat.grid');view(0,90);axis image;xlabel('x');ylabel('y');
caxis([0 1]);colorbar('vert');
title('Finite Difference Approximation to T(t,x,y)');

subplot(3,1,3);
X=[a1:h:a2]’;
Y=[b1:h:b2]’;
surf(X,Y,Sol);view(0,90);axis image;xlabel('x');ylabel('y');
caxis([0 1]);colorbar('vert');
title('Semi-Discrete Approximation to T(t,x,y)');

subplot(3,1,1);
func;
func.m

R = n + 2;
K = .3;

x = [0:h:pi]';
y = [-pi/2:h:pi/2]';

for i = 1:1:R;
    W(:,i) = cos(y(i,1)) * sin(x);
end;

W = exp(-K*tfinal) * W;
surf(x,y,W);view(0,90);axis image;caxis([0 1]);colorbar('vert');
xlabel('x');
ylabel('y');
title('The exact solution T(t,x,y)');
rhs.m

function y = rhs(t,w);

global A B N u n x y K;

[u1 u2 u3 u4] = controls(t);
u_initial=[u1;u2;u3;u4];

for k=1:1:N;
u(k,1) = u_initial(k,1);
end;

f = K * exp(-K * t) * exact(x,y);

y = sparse(A)*w + sparse(B)*u + f;
exact.m

function exa = exact(x,y);

global n;

f = [];

for i=1:1:n;
    w = cos(y(i,1)) * sin(x);
    f = [f;w];
end;

exa = f;
bi.m

%Code to construct the temperature profile functions

function [l1,l2,l3,l4] = bi(N,a1,a2,x);

if (x<(a1 + (a2-a1)/N));
    l1=abs(sin(((N*pi)/(a2-a1))*(x-a1)));
    else l1=0;
end;

if (x>=(a1 + (a2-a1)/N) & x<(a1 + 2*(a2-a1))/N);
    l2=abs(sin(((N*pi)/(a2-a1))*(x-a1)));
    else l2=0;
end;

if (x>=(a1 + 2*(a2-a1)/N) & x<(a1 + 3*(a2-a1))/N);
    l3=abs(sin(((N*pi)/(a2-a1))*(x-a1)));
    else l3=0;
end;

if (x>=(a1 + 3*(a2-a1)/N) & x<(a1 + 4*(a2-a1))/N);
    l4=abs(sin(((N*pi)/(a2-a1))*(x-a1)));
    else l4=0;
end;
controls.m

% code to determine the values of each u_k(t)
function [t1,t2,t3,t4] = controls(time);

global N;

heatmax1=1;
heatmax2=1;
heatmax3=1;
heatmax4=1;

tswitch1a = 4; % The time value the first lamp turns on.
tswitch2a = 4; % The time value the second lamp turns on.
tswitch3a = 3; % The time value the third lamp turns on.
tswitch4a = 3; % The time value the fourth lamp turns on.

if (time < tswitch1a) t1=0; end;
    if(time >= tswitch1a) t1=(time-tswitch1a); end;
    if((time-tswitch1a) >= heatmax1) t1=1; end;

if (time < tswitch2a) t2 = 0; end;
    if(time >= tswitch2a) t2=(time-tswitch2a); end;
    if((time-tswitch2a) >= heatmax2) t2=1; end;
    if(N<2) t2=0; end;

if (time < tswitch3a) t3=0; end;
    if(time >= tswitch3a) t3=(time-tswitch3a); end;
    if ((time-tswitch3a) >= heatmax3) t3 = 1; end;
    if(N<3) t3=0; end;

if (time < tswitch4a) t4=0; end;
    if(time >= tswitch4a) t4 = (time-tswitch4a); end;
    if ((time - tswitch4a) >= heatmax4) t4 = 1; end;
    if(N<4) t4=0; end;
Code to construct a movie matrix from a PDESolve control file

function M = evolve(filename)

    global MM;

    fp = fopen(filename, 'rt');

    index = 0;
    line = fgetl(fp);

    while (line ~= -1)
        index = index + 1;
        h=figure(index);
        clf reset;
        PDEview(line);axis image;
        view(0,90);
        caxis([0 1]);
        MM(:,index) = getframe;
        close(h);
        % Read next line.
        line = fgetl(fp);
    end;
R. Chris Camphouse was born in Rexburg, Idaho on September 30, 1972. He graduated from Madras High School, located in Madras, Oregon, in 1990. After completing high school, Chris received his undergraduate degree in mathematics at Oregon State University. While at OSU, he did an undergraduate research project with Dr. Belinda King, now of Virginia Tech. Dr. King introduced Chris to Dr. John Burns of Virginia Tech. Dr. Burns took Chris on as a graduate student.

In 1999, Chris received his Master of Science in mathematics from Virginia Tech. At the present time, Chris is enrolled in the Ph. D. program at Virginia Tech and is being mentored by Dr. John Burns.