High Order Collocation Software for the Numerical Solution of Fourth Order Parabolic PDEs

By
Ling Lin

A Thesis Submitted to
Saint Mary's University, Halifax, Nova Scotia
in Partial Fulfillment of the Requirements for
the Degree of Master of Science in Applied Science

Halifax, Nova Scotia
August, 2009

Your file Votre référence
ISBN: 978-0-494-55990-1
Our file Notre référence
ISBN: 978-0-494-55990-1

Certification

High Order Collocation Software for the Numerical Solution of Fourth Order Parabolic PDEs

by

Ling Lin

A Thesis Submitted to Saint Mary's University, Halifax, Nova Scotia,
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Applied Science

August 27, 2009

Examining Committee:

Approved: Dr. Ronald Haynes, External Examiner
Department of Mathematics and Statistics, Acadia University

Approved: Dr. Paul Muir, Co-Senior Supervisor
Department of Math and Computing Science

Approved: Dr. Patrick Keast, Co-Senior Supervisor
Department of Math and Statistics, Dalhousie University

Approved: Dr. Norma Linney, Supervisory Committee
Department of Math and Computing Science

Approved: Dr. Walt Finden, Supervisory Committee
Department of Math and Computing Science

Approved: Dr. Hai Wang, Program Representative

Approved: Dr. Pawan Lingras, Graduate Studies Representative

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisors, Dr. Paul Muir and Dr. Patrick Keast (Dalhousie University) for their patience, suggestions and assistances through out the whole process of doing the thesis. When I felt depressed because of the stagnation of the research, they gave me lots of encouragement and help me get out of the hard time. They showed me how to do a research and what the world of Numerical Analysis looks like. It is a great pleasure for me to complete this thesis under their supervision.

I also would like to thank my supervisory committees, Dr. Walt Finden (Saint Mary's University) and Dr. Norma Linney (Saint Mary's University), for their valuable suggestions on my thesis. I would like to thank my external examiner, Dr. Ronald Haynes (Acadia University), for his useful comments and suggestions.

Finally, I would like to express my appreciation to my father and mother for their love and encouragement. Except that, a lot of people who have offered help and support to me are not mentioned here, but I really appreciated them band I will remember all what you do for me.

Contents

List of Tables

List of Figures

Abstract

High Order Collocation Software for the Numerical

Solution of Fourth Order Parabolic PDEs

By Ling Lin

BACOL is an efficient software package for solving systems of second order parabolic PDEs in one space dimension. A significant feature of the package is that it employs adaptive error control in both time and space. A second order PDE depends on the solution, u, and its first and second derivatives, u_x and u_{xx}. However, many applications lead to mathematical models which involve fourth order PDEs. Fourth order PDEs depend on u, u_x, u_{xx}, u_{xxx}, and u_{xxxx}. One contribution of the thesis is that it provides a survey of applications in which fourth order PDEs arise.

The thesis focuses on how to extend BACOL so that it can handle fourth order PDEs. We have explored a somewhat novel approach that involves converting the fourth order PDE to a coupled system which contains one second order PDE and one second order ODE (in space). A careful investigation of the BACOL package is carried out in order to extend it so that it can treat this coupled PDE/ODE system directly; the new software is called BACOL42. For comparison purposes we have also considered an approximate form of the converted system that can be solved using the original BACOL software. Numerical results are provided to demonstrate the effectiveness of BACOL42. The thesis also provides a numerical study of two other PDE solvers, *pdepe* and MOVCOL4, that can be applied to solve fourth order PDEs.

August 27, 2009

Chapter 1

Introduction

A partial differential equation (PDE) is a type of equation which involves an unknown function of several independent variables and its partial derivatives with respect to those variables. It can describe how a physical quantity, e.g. heat, might change with respect to variables like time and space. PDEs are used to model many applications, including physical processes such as the diffusion of sound or heat, the growth in a population, the spread of a virus, etc.

PDE models are usually too complicated to be solved by hand. In most cases PDEs may not even have a known exact solution. Hence people use computer software to try to compute an approximate solution. If a software package returns an *approximate* solution, it is important that the error in that approximate solution be assessed. An important feature of some software packages is called *adaptive error control.* The concept of adaptive error control is to adapt the computation so that an estimate of the error of the numerical solution returned by the software is less than a user

provided error tolerance.

A common subclass of PDEs is time-dependent parabolic PDEs in one spatial dimension. Such problems have solutions that depend on time, t, and space, x; that is, the solution is typically written as $u(x,t)$. An important class of software packages for the solution of this class is based on an approach called the method of lines (MOL [8], [40]). While many software packages for this problem class attempt to control the error in time, only a few also attempt to control the error in space. Furthermore almost all available packages are designed for second order parabolic PDEs. A second order PDE is one in which the highest spatial derivative that appears is the second derivative, $u_{xx}(x,t)$.

BACOL [36] and BACOLR [37] are efficient software packages for solving systems of second order parabolic PDEs in one space dimension; a significant feature of these packages is that they employ adaptive error control in both time and space. For the test problems in [35], BACOL has been shown to be superior to many currently available software packages for solving second order PDEs. And in [37], BACOLR has been shown to be comparable to and in some cases superior to BACOL.

On the other hand, there are many applications in which mathematical models arise that involve fourth order PDEs, i.e., PDEs in which a $u_{xxxx}(x,t)$ term appears. BACOL/BACOLR are not designed to handle such problems. To our knowledge, only MOVCOL4 [33] and HP4 [45] can handle fourth order PDEs. MOVCOL4, to our knowledge the first MOL package designed to handle fourth order PDEs, is an extension of the well-known second order PDE solver MOVCOL [21]. MOVCOL4

only attempts to control the error in time; there is no attempt to control the error in space. The package uses a moving mesh approach to adapt the location of a given number of mesh points to the solution behavior and achieves about as much accuracy in space as can be obtained using the given number of mesh points. We will discuss MOVCOL4 later in this thesis. HP4 is a modification of the well-known second order parabolic PDE solver HPNEW [37]. HP4 attempts to control the spatial and temporal errors, and thus represents a significant contribution to software for the treatment of fourth order parabolic PDEs. However, we will not consider HP4 in the thesis as it was only very recently released.

In this thesis, we focus on solving one single time-dependent, fourth order PDE which has the general form:

$$u_t(x,t) = f(x,t,u,u_x,u_{xx},u_{xxx},u_{xxxx}), \quad a \le x \le b, \quad t \ge t_0, \tag{1.1}$$

supplemented with the initial conditions

$$u(x,t_0) = u_0(x), \quad a \le x \le b,$$

and separated boundary conditions

$$b_{L,1}(t, u(a,t), u_x(a,t), u_{xx}(a,t), u_{xxx}(a,t)) = 0, \quad t \geq t_0,$$

$$b_{L,2}(t, u(a,t), u_x(a,t), u_{xx}(a,t), u_{xxx}(a,t)) = 0, \quad t \geq t_0,$$

$$b_{R,1}(t, u(b,t), u_x(b,t), u_{xx}(b,t), u_{xxx}(b,t)) = 0, \quad t \geq t_0,$$

$$b_{R,2}(t, u(b,t), u_x(b,t), u_{xx}(b,t), u_{xxx}(b,t)) = 0, \quad t \geq t_0.$$

(The above equations specify two boundary conditions at each end point; this is the most common case and currently the only one the our software can handle; however, it would require only a minor generalization of the software to handle the remaining cases which involve one boundary condition at one end point and three at the other.)

We will assume the initial solution $u_0(x)$ is at least twice differentiable. (The algorithm we will discuss in this thesis requires that $u_0''(x)$ exist.)

Instead of considering the fourth order PDE directly, let $u_1(x,t) = u(x,t)$ and $u_2(x,t) = u_{xx}(x,t)$ and consider the following converted system,

$$\begin{cases} (u_1)_t = f(x,t,u_1,(u_1)_x,(u_1)_{xx},(u_2)_x,(u_2)_{xx}), \\ 0 = (u_1)_{xx} - u_2, \end{cases} \tag{1.2}$$

with the initial conditions

$$\begin{aligned} u_1(x,t_0) &= u_0(x), \quad a \le x \le b, \\ u_2(x,t_0) &= u_0''(x), \quad a \le x \le b, \end{aligned}$$

and separated boundary conditions,

$$\begin{aligned} b_{L,1}(t,u_1(a,t),(u_1)_x(a,t),u_2(a,t),(u_2)_x(a,t)) &= 0, \quad t \ge t_0, \\ b_{L,2}(t,u_1(a,t),(u_1)_x(a,t),u_2(a,t),(u_2)_x(a,t)) &= 0, \quad t \ge t_0, \\ b_{R,1}(t,u_1(b,t),(u_1)_x(b,t),u_2(b,t),(u_2)_x(b,t)) &= 0, \quad t \ge t_0, \\ b_{R,2}(t,u_1(b,t),(u_1)_x(b,t),u_2(b,t),(u_2)_x(b,t)) &= 0, \quad t \ge t_0. \end{aligned}$$

The converted system is obviously equivalent to the original fourth order PDE. The converted system is discussed in [33] but the authors do not consider the idea further because it leads to complications with the moving mesh approach they are considering.

The first equation of the converted system is a second order PDE, but the second equation of the converted system is a second order ODE (in space). BACOL can only solve systems of PDEs - not a coupled PDE/ODE system. One of the primary goals of this thesis is to investigate the possibility of modifying the original BACOL software in order to allow it to handle the above coupled PDE/ODE system. We will refer to the resultant software as BACOL42.

The modification of BACOL to allow it to solve a coupled PDE/ODE system required a very careful and thorough review of the BACOL package, followed by the application of a small but detailed set of modifications to the source code. (BACOL consists of approximately 51 subprograms and overall includes approximately 10000 lines of code written in Fortran77.)

During the development of BACOL42, it was useful to be able to compute approximate solutions to the converted PDE/ODE system for comparison purposes, and we realized that such solutions could be obtained by considering an approximate form of the converted PDE/ODE system:

$$\begin{cases} (u_1)_t = f(t, x, u_1, (u_1)_x, (u_1)_{xx}, (u_2)_x, (u_2)_{xx}), \\ \\ \epsilon(u_2)_t = (u_1)_{xx} - u_2, \end{cases} \tag{1.3}$$

with the boundary and initial conditions unchanged. It is clear that in the limit as $\epsilon \rightarrow 0$, the above problem approaches the converted system, and therefore it would appear to be the case that the solution of the above system would approach the solution of the converted system, as $\epsilon \rightarrow 0$, except possibly in a boundary layer,

assuming f and u and its derivatives are sufficiently well behaved. We will refer to the above system as the ϵ version of the original converted system and later in the thesis we will explore the approach of applying the original BACOL software to solve this system. Rewriting the ϵ version of the original problem as

$$\begin{cases} (u_1)_t = f(t, x, u_1, (u_1)_x, (u_1)_{xx}, (u_2)_x, (u_2)_{xx}), \\ (u_2)_t = \frac{1}{\epsilon}\left((u_1)_{xx} - u_2\right), \end{cases}$$

gives a second order PDE system in standard form and BACOL can therefore be directly applied to this system.

One contribution of the thesis is to provide a survey of applications in which fourth order PDEs arise. In Chapter 2, we discuss several models such as the Kuramoto-Sivashinsky equation [24], the extended Fisher-Kolmogorov equation [11], and the Cahn-Hilliard equation [6]. Also two simple fourth order PDEs with exact solutions are presented.

Chapter 3 provides a review of the underlying algorithms upon which BACOL and BACOLR are based; this includes Runge-Kutta methods, Backward Differentiation Formulas, B-splines, and collocation. In this chapter, we also discuss several other currently available software packages for second order PDEs, namely, HPNEW, *pdepe* [29], and MOVCOL. In addition, we also discuss MOVCOL4, a package mentioned earlier, for the solution of fourth order PDEs.

In Chapter 4, the modifications of the original BACOL package in order to allow it to handle the converted system are described in detail. Another contribution of this

thesis is that it explores the initial steps required to construct a version of BACOL that can compute numerical solutions to general classes of coupled PDEs and ODEs (in space). The development of BACOL42 assumes that the ODE has the form $0 = (u_1)_{xx} - u_2$ but the approach considered in this thesis would appear to be applicable to more general forms of ODEs.

We consider five numerical examples and present detailed performance for BACOL42 on these problems in Chapter 5. For comparison purposes, we also consider the alternative approach, described earlier, of applying the original BACOL software to the ϵ version of each problem. Neither *pdepe* nor MOVCOL4 consider spatial error control and the original BACOL software is only applied to an approximate form of the given problem; it is only the BACOL42 code (and the recently developed HP4 code, mentioned earlier) that are able to compute a numerical solution to a fourth order PDE, with an attempt to control both the spatial and temporal errors with respect to a given user-provided tolerance.

Another contribution of the thesis comes in Chapter 6, where we provide numerical results to explore the effectiveness of two other software packages applied to solve fourth order PDEs. We present results with (i) *pdepe* applied to the converted system version of each problem, and (ii) MOVCOL4 applied directly to the original fourth order version of the problem.

In the final chapter, we give our conclusions and also provide some suggestions for future work.

Chapter 2

Review of Applications Involving Fourth order PDEs

In this chapter, we provide some examples of fourth order parabolic PDEs. In [33], the authors explain that many complex real-world applications require PDE models that include fourth order terms in order to better capture the behavior of the quantities being modeled. Section 2.1 discusses the first example which is from a cell biology application. The Quantum Drift Diffusion model is given in Section 2.2. In Section 2.3, we introduce a generalized form of the longwave unstable thin film equation and also give three other applications including the Kuramoto-Sivashinsky equation, an application describing droplet breakup in a Hele-Shaw cell, and another application modeling a van der Waals rupture of a thin film. The fourth application, in Section 2.4, is the extended Fisher-Kolmogorov equation. In Section 2.5, we consider some equations modeling image denoising and segmentation. Section 2.6 discusses

the Cahn-Hilliard equation. Finally, in Section 2.7 we give two simple fourth order PDEs with exact solutions that will be employed as test problems later in the thesis.

Notation

The following symbols are used when we display the equations associated with the applications we considered in this chapter:

- Let $u(\underline{x}) = u(x_1, x_2, \cdots, x_n)$ be a (four times) differentiable function on a given domain; we have $u : \Re^n \rightarrow \Re$.

- Let $\underline{F}(\underline{x}) = [F_1(\underline{x}), F_2(\underline{x}), \cdots, F_n(\underline{x})]^T$ be a differentiable vector function on a given domain; we have $\underline{F}(\underline{x}) : \Re^n \rightarrow \Re^n$.

- $\dfrac{\partial u(\underline{x})}{\partial x_i}$, represents the derivative of $u(\underline{x})$ with respect to x_i.

- $\bigtriangledown u(\underline{x}) = \left(\dfrac{\partial u(\underline{x})}{\partial x_1}, \dfrac{\partial u(\underline{x})}{\partial x_2}, \cdots, \dfrac{\partial u(\underline{x})}{\partial x_n} \right)^T$ is the gradient of $u(\underline{x})$.

- $div\underline{F}(\underline{x}) = \dfrac{\partial F_1(\underline{x})}{\partial x_1} + \dfrac{\partial F_2(\underline{x})}{\partial x_2} + \cdots + \dfrac{\partial F_n(\underline{x})}{\partial x_n}$ is the divergence of $\underline{F}(\underline{x})$.

- $\bigtriangleup u(\underline{x}) = div \nabla u(\underline{x}) = \nabla^2 u(\underline{x}) = \sum_{i=1}^{n} \dfrac{\partial^2 u(\underline{x})}{\partial x_i^2}$ is the Laplacian of $u(\underline{x})$.

2.1 Cell Biology Problem

This description is taken from [7]. Within the nucleus of a cell, splicing factors (SFs) are nuclear proteins that remove introns (non-coding gene sequences) from precursor mRNA molecules to form the mature mRNA molecules that will be transported to the cytoplasm - the part of a cell including the region between the cell membrane and the nucleus. The SFs are in continuous flux between membraneless clusters known as speckles and individual SFs distributed throughout the nucleoplasm region, i.e., the nucleus of the cell. The SFs move randomly throughout the cell nucleus. They are heterogeneously distributed in the nuclei of eukaryotic cells that are enriched in pre-mRNA SFs - see Figure 2.1. This is an indirect immunofluorescence image of the speckled distribution of the SFs within the cell nucleus.

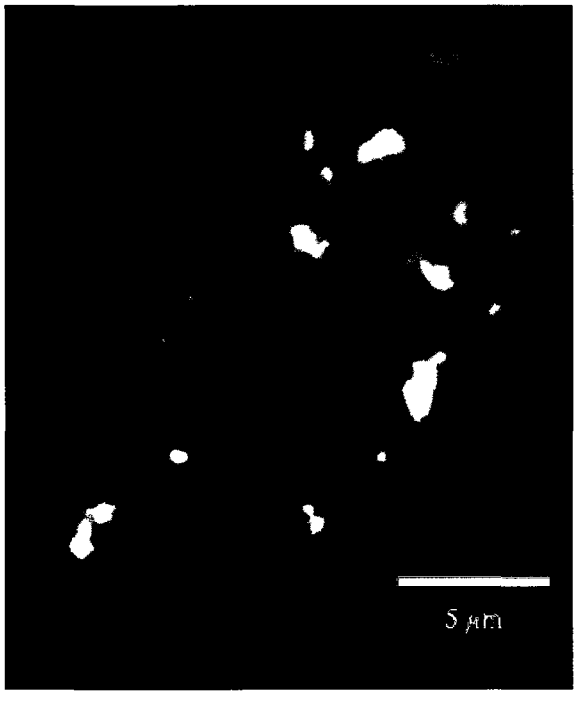

Figure 2.1: The Distribution of SFs in the Nucleus [7].

Phosphorylation is the process of adding phosphate (PO_4) groups to a molecule or an organic compound while dephosphorylation is the opposite process (see [43, 44]). ρ is the rate of phosphorylation while δ is the rate of dephosphorylation. If the distance between SFs is larger than the range of influence of self-interaction, σ, the above processes will not happen. This model considers $u(x,t)$, the density of phosphorylated, i.e., clustered, SFs in the cell nucleus, and $v(x,t)$ the density of dephosphorylated, i.e., individual, SFs in the cell nucleus.

The model is based on two hypotheses: (1) that self-organization is responsible for the formation and disappearance of speckles, which are modulated by phosphorylation and dephosphorylation, and (2) that the existence of an underlying nuclear structure, i.e., a nuclear scaffold, is a major factor in the organization of SFs.

The effect of phosphorylation and dephosphorylation on the SFs can be represented as in Figure 2.2. In this figure, the thin arrows represent that splicing factors are tied to a nuclear scaffold or nuclear matrix. The thick arrows represent that the phosphorylation and dephosphorylation processes modulate the continuous flux between the speckles and the nucleoplasm. Dephosphorylation leads to self-organization of the SFs into speckles, whereas phosphorylation leads to the opposite process.

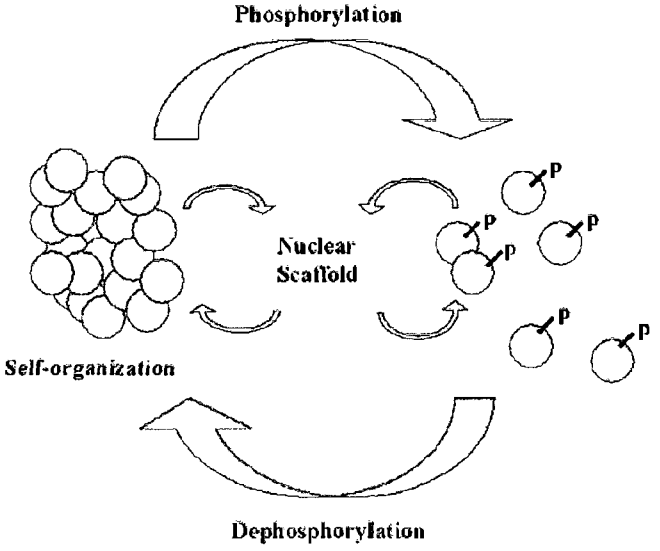

Figure 2.2: The Effect of Phosphorylation and Dephosphorylation on the SFs [7].

The corresponding model, a fourth order aggregation-diffusion PDE, has the form,

$$\begin{cases} \dfrac{\partial v(x,t)}{\partial t} = \dfrac{\partial^2 v(x,t)}{\partial x^2} - \delta v(x,t) + \rho u(x,t), \\ \dfrac{\partial u(x,t)}{\partial t} = \dfrac{\partial}{\partial x}[(1-u(x,t))\dfrac{\partial u(x,t)}{\partial x}] - \dfrac{\partial}{\partial x^2}[\dfrac{\sigma^2}{12}u(x,t)\dfrac{\partial^2 u(x,t)}{\partial x^2}] \\ \qquad\qquad +\delta v(x,t) - \rho u(x,t), \end{cases} \tag{2.1}$$

with the no-flux boundary conditions:

$$\begin{aligned} &\frac{\partial v}{\partial x}(0,t) = \frac{\partial v}{\partial x}(1,t) = \frac{\partial u}{\partial x}(0,t) = \frac{\partial u}{\partial x}(1,t) = 0, \\ &\frac{\partial^3 u}{\partial x^3}(0,t) = \frac{\partial^3 u}{\partial x^3}(1,t) = 0, \end{aligned}$$

Rewriting Equation (2.1), we get

$$\begin{cases} v_t(x,t) = v_{xx}(x,t) - \delta v(x,t) + \rho u(x,t), \\ u_t(x,t) = -\frac{\sigma^2}{12}(u(x,t)u_{xxxx}(x,t) + 2u_x(x,t)u_{xxx}(x,t) + u_{xx}^2(x,t)) \\ \qquad\qquad +(1-u(x,t))u_{xx}(x,t) - u_x^2(x,t) + \delta v(x,t) - \rho u(x,t), \end{cases} \tag{2.2}$$

which is a system of PDEs combining one second order PDE and one fourth order PDE. Initial conditions are chosen by considering small random perturbations of the steady state solution to (2.2).

2.2 Quantum Drift Diffusion (QDD) Model

The second application is the Quantum Drift Diffusion (QDD) model which describes the transport of quantum charge-species in strong interaction within a surrounding medium at a given temperature. It has been formally derived by Degond, Mehats, and Ringhofer [12] from a collisional Wigner equation (see the definition of notation on Page 9),

$$\begin{cases} n_t(\underline{x},t) + divJ(\underline{x},t) = 0, \\ J(\underline{x},t) + T\nabla n(\underline{x},t) - n(\underline{x},t)\nabla V(\underline{x},t) - \frac{\varepsilon^2}{2}n(\underline{x},t)\nabla(\frac{\Delta\sqrt{n(\underline{x},t)}}{\sqrt{n(\underline{x},t)}}) = 0, \end{cases} \tag{2.3}$$

where

$n(\underline{x},t)$ is the particle density,

$J(\underline{x}, t)$ is the current density,

$V(\underline{x}, t)$ is the electrostrostatic potential,

ε is the scaled Planck constant, and

T is the scaled temperature.

After setting the temperature, T, and the electrostrostatic potential, $V(\underline{x}, t)$, to zero, (2.3) can be rewritten as a fourth order parabolic PDE for the particle density, $n(\underline{x}, t)$ [18]:

$$n_t(\underline{x}, t) = -div\left(\frac{\varepsilon^2}{2} n(\underline{x}, t) \bigtriangledown \left(\frac{\triangle\sqrt{n(\underline{x}, t)}}{\sqrt{n(\underline{x}, t)}}\right)\right). \tag{2.4}$$

In one space dimension, a simpler form of this equation is often considered; it is

$$n_t(x, t) + \frac{\varepsilon^2}{2}(n(x, t)(\log n(x, t))_{xx})_{xx} = 0. \tag{2.5}$$

The paper [22] discusses the analytical solution of (2.5) with Dirichlet and Neumann boundary conditions:

$$n(0, t) = n(1, t) = 1,$$

$$n_x(0, t) = n_x(1, t) = 0.$$

2.3 Thin Film Equations

The paper [39] introduces the longwave unstable generalized thin film equation:

$$h_t(x,t) = -(h^m(x,t)h_x(x,t))_x - (h^n(x,t)h_{xxx}(x,t))_x, \tag{2.6}$$

where

$h(x,t)$ is the height of the evolving free-surface,

m is the power of the destabilizing second order diffusive term,

n is the power of the stabilizing fourth order diffusive term.

The paper [41] discusses the existence and uniqueness of solutions for (2.6).

Some special cases of (2.6) are well-known. When $n = 0, m = 1$, it becomes a modified Kuramoto-Sivashinsky equation; when $n = 1, m = 1$, it describes droplet breakup in a Hele-Shaw cell; when $n = 3, m = -1$, it gives the van der Waals rupture of a thin film.

2.3.1 The Kuramoto-Sivashinsky Equation

One form of the Kuramoto-Sivashinsky equation [24]((2.6) with $n = 0$ and $m = 1$) is:

$$h_t(x,t) = -h_{xxxx}(x,t) - h_{xx}(x,t) - \frac{1}{2}h_x^2(x,t),$$

where $h(x,t)$ is the position of the interface between the solid and liquid phases.

Introducing a periodic boundary condition gives the periodic Kuramoto-Sivashinsky equation (the period length is L)

$$h_t(x,t) = -h_{xxxx}(x,t) - h_{xx}(x,t) + h_x^2(x,t), \qquad h(x+L) = h(x),$$

which arises in modeling the dynamics of a premixed flame (see [15]) and the dynamics of solidification (see [16]).

Another form of the Kuramoto-Sivashinsky equation [1] is

$$u_t = -uu_x - u_{xx} - u_{xxxx}, \quad x \in [0, 32\pi.] \tag{2.7}$$

The initial condition is

$$u(x,0) = \cos\left(\frac{x}{16}\right)\left(1 + \sin\left(\frac{x}{16}\right)\right).$$

The modified Kuramoto-Sivashinsky equation

$$h_t(x,t) = -h_{xxxx}(x,t) - h_{xx}(x,t) + (1-\lambda)h_x(x,t)^2 + \lambda h_{xx}^2(x,t),$$

is studied as a one-dimensional model for the dynamics of a hypercooled melt in [34].

2.3.2 Droplet Breakup in a Hele-Shaw Cell

This application is from the model of the flow in a Hele-Shaw cell (see [9]). A Hele-Shaw cell contains two unmixable fluids which are divided by an interface. The paper discusses whether a thin neck between two masses of fluid can develop, get thinner, and finally break. Figure 2.3 shows a special case of the problem where the neck is assumed to be symmetrical. $h(x,t)$ is the neck region, x is the distance along the symmetry line and t is the time. L is assumed to be much greater than W.

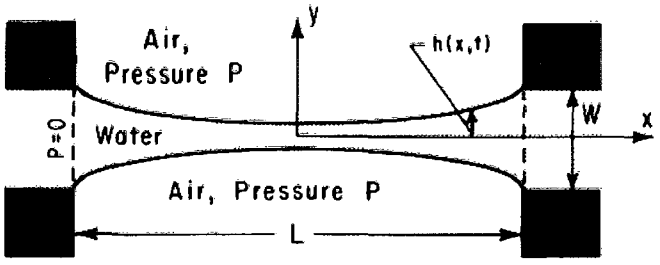

Figure 2.3: a Special Case of the Flow in a Hele-Shaw Cell [9].

Letting L be 2, the equation has the following form:

$$h_t(x,t) + (h(x,t)h_{xxx}(x,t))_x = 0, \quad -1 \leq x \leq 1, \tag{2.8}$$

with the initial condition:

$$h(x,0) = 1,$$

and the boundary conditions:

$$h(\pm 1, t) = 1,$$

$$h_{xx}(\pm 1, t) = -p,$$

where p is a nonzero pressure. Techniques for the numerical solution of (2.8) are discussed in the paper [2].

2.3.3 van der Waals Rupture of a Thin Film

This equation is associated with the modeling of a thin film (see [38]). The model represents the process through which a thin liquid film on a solid substrate overcomes surface tension and ruptures because of van der Waals forces.

The non-dimensional form of the model is derived based on the assumption that there is no slip at the solid substrate and free slip at the thin surface; the governing equation is

$$\frac{\partial h(r,t)}{\partial t} + \frac{1}{r}\frac{\partial}{\partial r}\left[rh^3(r,t)\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial h(r,t)}{\partial r}\right)\right) + \frac{r}{h(r,t)}\frac{\partial h(r,t)}{\partial r}\right] = 0, \qquad (2.9)$$

with the initial condition:

$$h(r,0) = \frac{10}{9}\left[1 - \frac{1}{10}\cos\left(\frac{2\pi r}{\lambda}\right)\right],$$

and the boundary conditions:

$$h_r(0,t) = 0, h_r\left(\frac{\lambda}{4}, t\right) = \frac{\lambda}{4},$$
$$h_{rrr}(0,t) = 0, h_{rrr}\left(\frac{\lambda}{4}, t\right) = \frac{\lambda}{4},$$

where $h(r,t)$ is the thickness of the thin liquid, r is the radial coordinate and λ is a problem dependent parameter..

2.4 Extended Fisher-Kolmogorov (EFK) Equation

This equation is called the extended Fisher-Kolmogorov (EFK) equation which arise in the discussion of the propagation of fronts into unstable states of a bistable system; it was proposed by Dee and van Saarloos in the paper [11]:

$$u_t(x,t) = -\gamma u_{xxxx}(x,t) + u_{xx}(x,t) + u(x,t) - u^3(x,t), \tag{2.10}$$

where $-\infty < x < \infty$ and $\gamma > 0$. The initial condition is:

$$u(x,0) = 0.1e^{-x^2}, \gamma = 0.03,$$

and the boundary conditions are:

$$\lim_{x\to\infty} u(0,x) = 1,$$
$$\lim_{x\to-\infty} u(0,x) = -1.$$

At the same time, Coullet, Elphick and Repaux also obtained the EFK equation in the study of the spatial complexity of one-dimensional patterns (see [10]).

2.5 Applications in Image Processing

Applications involving fourth order PDEs are widely used in image processing. In [4], Bertozzi and Greer study a model of image denoising and segmentation; it has the form

$$u_t(\underline{x},t) = -\nabla \cdot (g(s) \triangle u(\underline{x},t) \triangle \nabla u(\underline{x},t)) + \lambda(f(\underline{x},t) - u(\underline{x},t)), \tag{2.11}$$

where

$f(\underline{x},t)$ is a noisy signal,

$u(\underline{x},t)$ is the image intensity,

$g(s) = k^2/(k^2+s^2)$ is a curvature threshold, k, s are parameters,

λ is a fidelity-matching parameter.

Lysaker *et al.* [27] also use a fourth order PDE in order to model the removal of noise from digital images. The model has the form:

$$\begin{aligned} u_t(\underline{x},t) = & - \left(\frac{u_{xx}(\underline{x},t)}{|D^2u(\underline{x},t)|}\right)_{xx} - \left(\frac{u_{xy}(\underline{x},t)}{|D^2u(\underline{x},t)|}\right)_{yx} - \left(\frac{u_{yx}(\underline{x},t)}{|D^2u(\underline{x},t)|}\right)_{xy} \\ & - \left(\frac{u_{yy}(\underline{x},t)}{|D^2u(\underline{x},t)|}\right)_{yy} - \lambda\left(u(\underline{x},t) - u_0(\underline{x},t)\right), \end{aligned}$$

with the boundary conditions:

$$\left(\frac{u_{xx}(\underline{x}_L,t)}{|u_{xx}(\underline{x}_L,t)|}\right)n_1+\left(\frac{u_{yy}(\underline{x}_L,t)}{|u_{yy}(\underline{x}_L,t)|}\right)n_2=0,$$
$$\left(\frac{u_{xx}(\underline{x}_R,t)}{|u_{xx}(\underline{x}_R,t)|}\right)_x n_1+\left(\frac{u_{yy}(\underline{x}_R,t)}{|u_{yy}(\underline{x}_R,t)|}\right)_y n_2=0,$$

where

$|D^2u| = (|u_{xx}|^2+|u_{xy}|^2+|u_{yx}|^2+|u_{yy}|^2)^{1/2}$,

$u(\underline{x},t)$ is a digital image measured by the pixel intensity value, $\underline{x}=\begin{bmatrix} x \\ y \end{bmatrix}$,

$u_0(\underline{x})$ is an observation with random noise,

λ is a known parameter,

$\mathbf{n}=(n_1,n_2)^T$ is the outward normal direction on $\partial\Omega$,

$\Omega=[a,b]\times[c,d]$, $\underline{x}_L=\begin{bmatrix} a \\ c \end{bmatrix}$, $\underline{x}_R=\begin{bmatrix} b \\ d \end{bmatrix}$.

The existence and uniqueness of a solution to the following equation which represents a noise removal model is discussed in [26]:

$$\frac{\partial u(x,t)}{\partial t}+\frac{\partial^2}{\partial x^2}\Phi'\left(\frac{\partial^2 u(x,t)}{\partial x^2}\right)=0, \quad (x,t)\in Q_T,$$
$$u(0,t)=u(1,t)=u_x(0,t)=u_x(1,t)=0, \quad t\in(0,T),$$
$$u(x,0)=u_0(x), \quad x\in I,$$

where $I=(0,1)$, $Q_T=I\times(0,T)$, $u(x,t)$ is the image intensity, $\Phi:\Re\to\Re^+$ is a known function which is even, continuous, and convex with $\Phi>0$ for $t>0$.

2.6 Cahn-Hilliard Equation

This equation was first introduced by Cahn and Hilliard in 1958 in the paper [6]. It describes the process of phase separation in which two immiscible fluids become separated as time increases. It is applied in many fields, including material science and engineering (see [42]). It has the form:

$$u_t(x,t) = D(u^3(x,t) - u(x,t) - \gamma u_{xx}(x,t))_{xx},$$

where

$u(x,t)$ is the concentration of the fluid,

D is a diffusion coefficient,

$\sqrt{\gamma}$ is the length of the transition regions between the domains.

In [14], the authors discuss the solution of the convective Cahn-Hilliard (CCH) equation with periodic boundary conditions

$$u_t(x,t) = u(x,t)u_x(x,t) - (u(x,t) + u^3(x,t) - u_{xx}(x,t))_{xx}, \quad x \in (0,L), \quad t > 0,$$

$$u(x+L,t) = u(x,t), \quad x \in \Re, t > 0,$$

$$u(x,0) = u_0(x), \quad x \in \Re.$$

2.7 Some Simple Fourth Order PDEs

From the previous sections, we can see that fourth order PDEs arise in many fields. However, an exact solution is not usually known. Here we will give two simple fourth-order PDEs with exact solutions. The exact solution to the first equation is obtained by inspection since the equation is very simple. The exact solution to the second equation is provided in [33]. These equations have been useful during the development and testing of our software.

2.7.1 Example One

The first example is a simple Fourth order linear PDE:

$$u_t(x,t) = -u_{xxxx}(x,t), \qquad 0 \le x \le \pi, \qquad t > 0, \tag{2.12}$$

with the boundary conditions

$$u(0,t) = u(\pi,t) = 0, \qquad t > 0,$$

$$u_{xx}(0,t) = u_{xx}(\pi,t) = 0, \qquad t > 0,$$

and the initial condition

$$u(x,0) = \sin x, \qquad 0 \le x \le \pi.$$

The exact solution (see Figure 2.4) is

$$u(x,t) = e^{-t}\sin x.$$

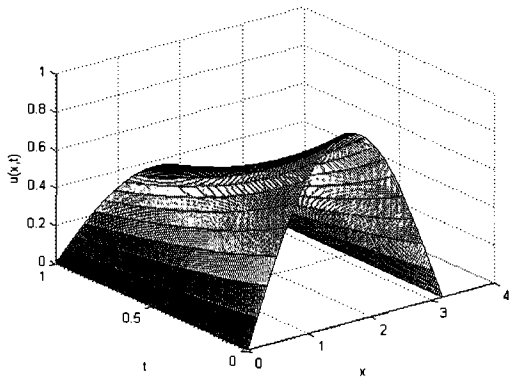

Figure 2.4: The Exact Solution of Equation (2.12)

2.7.2 Example Two

The second example is a test problem taken from [33]:

$$u_t(x,t) = -\left(\frac{\sin t}{3+\cos t}\right) u_{xxxx}(x,t), \qquad 0 \le x \le \pi, \qquad t > 0, \tag{2.13}$$

with the initial condition

$$u(x,0) = 1.2\cos x, \qquad 0 \le x \le \pi,$$

and the boundary conditions

$$u(0,t) = u_{xx}(\pi,t) = 0.3(\cos(t)+3),$$

$$u(\pi,t) = u_{xx}(0,t) = -0.3(\cos(t)+3).$$

The exact solution (see Figure 2.5) is

$$u(x,t) = 0.3(\cos t + 3)\cos x.$$

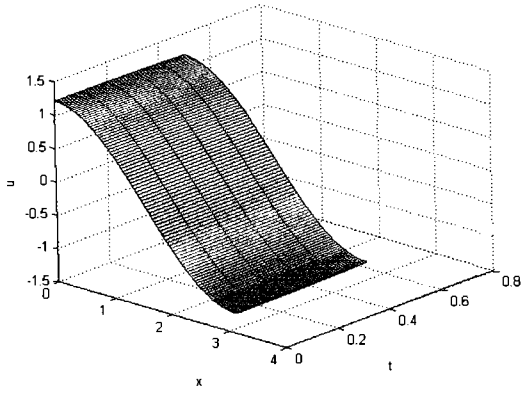

Figure 2.5: The Exact Solution of Equation (2.13)

Chapter 3

Review of Numerical Methods and Software Packages

Many software packages for the numerical solution of PDEs are based on an approach called the method of lines (MOL). The MOL is one of the most important computational approaches for solving PDEs. The MOL for solving a PDE proceeds in two separate steps:

- The spatial discretization: This involves partitioning the spatial domain into subintervals with a set of mesh points and approximating the spatial derivatives on each subinterval in some way, thereby replacing the PDEs by a system of ordinary differential equations (ODEs).

- The time integration: This involves using a software package to solve the ODEs and the boundary conditions. A system of ODEs coupled with a system of algebraic equations (in our case, the boundary conditions) is known as a system of differential-algebraic equations (DAEs). There are many good quality ODE/DAE solvers - we will consider a few later in this chapter.

3.1 Review of Numerical Methods

In sections 3.1.1 and 3.1.2 we discuss two numerical methods which provide the basis for software for the numerical solution of ODEs and DAEs. Section 3.1.3 introduces spline software which is used widely in the representation of numerical solutions. In Section 3.1.4, we describe one numerical method which is often employed as the spatial discretization process for the numerical solution of PDEs.

3.1.1 Runge-Kutta Methods

Runge-Kutta methods are important single-step methods for solving ODEs [17]. Single-step means that the current value can be obtained assuming only one previous solution approximation is given. The classical fourth order Runge-Kutta method is one of the most commonly used Runge-Kutta methods. Let us consider the following initial value problem:

$$y' = f(t, y), \quad y(t_0) = y_0.$$

Then, given a solution approximation, y_n, at t_n, we can obtain an approximation to the solution at t_{n+1}, which we call y_{n+1}, using the following equations:

$$y_{n+1} = y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right),$$

where

$$\begin{aligned}
h &= t_{n+1} - t_n, \\
k_1 &= f\left(t_n, y_n\right), \\
k_2 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right), \\
k_3 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right), \\
k_4 &= f\left(t_n + h, y_n + k_3\right).
\end{aligned}$$

The coefficients of this method can be written in a table, which is called a *Butcher tableau*. See Table 3.1.

Table 3.1: The Butcher Tableau for the Classical Fourth Order Runge-Kutta Method [17]

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

This classical fourth order Runge-Kutta method has many advantages. First, it

is *explicit.* That is, y_{n+1} can be obtained through an explicit calculation. In addition the accuracy of the approximate solution is reasonably high. If we assume that the solution at t_n is exact then then the *local* error in the solution at t_{n+1} is $O(h^5)$ for this method. The global error is then $O(h^4)$ so the method is fourth order over the entire time integration. It is easy to change the step size during the process since the Runge-Kutta method is a single-step method. However, each step requires four evaluations of f. Because the stability region is limited - see, e.g. [25], a small time step h needs to be chosen in order for the method to be stable. (A method is stable if the error does not grow as the time stepping proceeds.)

A modified version of the DAE/ODE solver RADAU5 (see [17, 19]) is employed in BACOLR to solve the DAEs that arise. RADAU5 is based on one particular Runge-Kutta method, Radau IIA. Using this method, we can obtain an approximate solution, y_{n+1}, at t_{n+1} from the following equations:

$$y_{n+1} = y_n + \frac{h}{36}\left((16-\sqrt{6})k_1 + (16+\sqrt{6})k_2 + 4k_3\right),$$

where

$$\begin{aligned}
h &= t_{n+1} - t_n, \\
k_1 &= f\left(t_n + \frac{4-\sqrt{6}}{10}h, y_n + \frac{88-7\sqrt{6}}{360}k_1 + \frac{296-169\sqrt{6}}{1800}k_2 + \frac{-2+3\sqrt{6}}{225}k_3\right), \\
k_2 &= f\left(t_n + \frac{4+\sqrt{6}}{10}h, y_n + \frac{296+169\sqrt{6}}{1800}k_1 + \frac{88+7\sqrt{6}}{360}k_2 + \frac{-2-3\sqrt{6}}{225}k_3\right), \\
k_3 &= f\left(t_n + h, y_n + \frac{16-\sqrt{6}}{36}k_1 + \frac{16+\sqrt{6}}{36}k_2 + \frac{1}{9}k_3\right).
\end{aligned}$$

We note that the k_1, k_2, k_3 values are defined implicitly (i.e., in terms of each other) and when f is nonlinear an iterative technique such as Newton's method must be employed to compute these values. On a single step the error is $O(h^6)$. The stability region of an implicit Runge-Kutta method is much larger than that of an explicit Runge-Kutta method. In fact the stability region typically includes the entire left half of the complex plane which implies that the stepsize is not restricted at all by stability considerations.

The coefficients of this method can be written in the form of the *Butcher tableau* given in Table 3.2.

Table 3.2: The Butcher Tableau for RADAU IIA [20]

$\frac{4-\sqrt{6}}{10}$	$\frac{88-7\sqrt{6}}{360}$	$\frac{296-169\sqrt{6}}{1800}$	$\frac{-2+3\sqrt{6}}{225}$
$\frac{4+\sqrt{6}}{10}$	$\frac{296+169\sqrt{6}}{1800}$	$\frac{88+7\sqrt{6}}{360}$	$\frac{-2-3\sqrt{6}}{225}$
1	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$
	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$

3.1.2 Backward Differentiation Formulas

Another DAE/ODE solver commonly employed by software (such as BACOL, HP-NEW, MOVCOL) for the numerical solution of PDEs is called DASSL (see [32]). DASSL is based on a family of numerical methods for the solution of DAEs/ODEs known as Backward Differentiation Formulas (BDFs). The BDFs represent one of the most useful families of linear multi-step methods; see, e.g., [25]. A multi-step method

uses several previous solution values in order to compute the current approximation. For the ODE, $y'(t) = f(t, y)$, a k-step BDF is defined by the formula

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h\beta_k f_{n+k},$$

where h is the fixed time step and α_j, $j = 1, 2, \cdots, k$ and β_k are given coefficients and y_{n+j}, $j = 0, 1, \cdots, k-1$ are known previous solution values; see Table 3.3. It is possible to choose the coefficients so that the order of the k-step BDF is k. The sizes of the regions of absolute stability of the BDF are larger than those of the same order of explicit Runge-Kutta method, for $k = 1, 2, ..., 6$. Furthermore, when $k = 1, 2$, the stability region includes the whole negative half-plane, in which case there are no stepsize restriction due to stability.

Table 3.3: Coefficients of BDF [25]

k	α_0	α_1	α_2	α_3	α_4	α_5	α_6	β_k
1	-1	1						1
2	$\frac{1}{3}$	$-\frac{4}{3}$	1					$\frac{2}{3}$
3	$-\frac{2}{11}$	$\frac{9}{11}$	$-\frac{18}{11}$	1				$\frac{6}{11}$
4	$\frac{3}{25}$	$-\frac{16}{25}$	$\frac{36}{25}$	$-\frac{48}{25}$	1			$\frac{12}{25}$
5	$-\frac{12}{137}$	$\frac{75}{137}$	$-\frac{200}{137}$	$\frac{300}{137}$	$-\frac{300}{137}$	1		$\frac{60}{137}$
6	$\frac{10}{147}$	$-\frac{72}{147}$	$\frac{225}{147}$	$-\frac{400}{147}$	$\frac{450}{147}$	$-\frac{360}{147}$	1	$\frac{60}{147}$

Using the above formula we can obtain the approximate solution at t_{n+k} called y_{n+k}. As an example we take $k = 5$, that is a $5 - step$ BDF. The formula becomes:

$$\begin{aligned} \alpha_0 y_n + \alpha_1 y_{n+1} \quad &+ \quad \alpha_2 y_{n+2} + \alpha_3 y_{n+3} + \\ \alpha_4 y_{n+4} \quad &+ \quad \alpha_5 y_{n+5} = h\beta_5 f(x_{n+5}, y_{n+5}). \end{aligned}$$

Substituting the coefficients for this formula from Table 3.3, we get:

$$\begin{aligned} -\frac{12}{137} y_n + \frac{75}{137} y_{n+1} \quad &- \quad \frac{200}{137} y_{n+2} + \frac{300}{137} y_{n+3} - \\ \frac{300}{137} y_{n+4} \quad &+ \quad y_{n+5} = h\frac{60}{137} f(x_{n+5}, y_{n+5}). \end{aligned}$$

We solve this formula for y_{n+5}, assuming previous solution values y_{n+4}, y_{n+3}, y_{n+2}, y_{n+1} and y_n. As mentioned earlier, when f is nonlinear, y_{n+5} must be computed using, for example, Newton's method.

From the above formula, we can see that a multi-step method requires only one new function value per step (although more function values may be necessary in the Newton iteration). However, it cannot start automatically. That is, the value of y_{n+5} cannot be obtained until five initial values are given. For example in the $5-step$ BDF, $y_n, y_{n+1}, y_{n+2}, y_{n+3}, y_{n+4}$ must be given in order to calculate y_{n+5}. Changing step size is more expensive than for a one-step method. A popular strategy for changing the stepsize involves using interpolation to obtain the values at the required past steps.

3.1.3 B-splines

In BACOL and BACOLR, the approximate solution is expressed as a piecewise polynomial in x with time dependent coefficients. The piecewise polynomial can be represented in terms of a B-spline basis (see [5]). Given $NINT+1$ knots, x_i, which divide the domain into $NINT$ pieces with

$$a = x_0 < x_1 < \cdots < x_{NINT} = b,$$

we then define a piecewise polynomial of order k (degree $k-1$) on each subinterval. That is, on each subinterval the segment of the piecewise polynomial has k coefficients. There are $NINT$ subintervals in the problem domain, therefore $k*NINT$ coefficients in total. The polynomials are $C^1 - continuous$, which means that the B-spline is continuous and differentiable at the mesh points $x_i, i = 1, \cdots, NINT-1$. Imposing these conditions leads to $2*(NINT-1)$ constraints. In order to solve for the $k*NINT$ coefficients,

$$NC = k * NINT - 2 * (NINT - 1) = NINT * (k-2) + 2$$

additional constraints are needed. Here NC is the total number of collocation conditions, including the boundary conditions, that will be applied (We will explain in the next subsection what we mean by collocation conditions). These conditions, together with the continuity conditions, give a total number of conditions that is equal to the number of free coefficients. The PDE solution $u(x,t)$ is approximated by $U(x,t)$

which can be expressed in terms of B-splines as:

$$U(x,t) = \sum_{i=1}^{NC} y_i(t) B_i(x), \qquad x \in [x_0, x_{NINT}],$$

where $y_i(t)$ are unknown time-dependent coefficients and $B_i(x)$ are the B-spline basis polynomials. Each $B_i(x)$ will be non-zero only in a relatively small region of the problem domain. See [5] for further details about the B-spline basis functions.

3.1.4 Collocation

In PDE software packages such as BACOL and BACOLR (as well as PDECOL [28] and EPDCOL [23]), that express the approximate solution in terms of B-splines, the spatial discretization is performed using a technique known as collocation. Suppose that the PDE has the following form:

$$u_t = f(t, x, u, u_x, u_{xx}), \qquad a \leq x \leq b,$$

with boundary conditions

$$b_L(t, u, u_x) = 0, \quad at\ x = a,$$

$$b_R(t, u, u_x) = 0, \quad at\ x = b.$$

The collocation method requires the approximate solution to exactly satisfy the PDE at a given set of collocation points, $\xi_l, l = 2, \cdots, NC-1$, and at the boundary points

($\xi_1 = a$ and $\xi_{NC} = b$). That is, we require $U(x,t)$ to satisfy

$$0 = b_L(t, U(0,t), U(0,t)_x),$$

$$U_t(\xi_l, t) = f(t, x, U(\xi_l, t), U(\xi_l, t)_x, U(\xi_l, t)_{xx}),\ l = 2, \cdots, NC-1,$$

$$0 = b_R(t, U(1,t), U(1,t)_x).$$

Substituting for $U_t(\xi_l, t)$ with $\Sigma_{i=1}^{NC} y_i'(t) B_i(x)$, we then obtain a system of differential equations coupled with two algebraic equations. The DAE system has the form,

$$b_L(t, U(0,t), U(0,t)_x) = 0,$$

$$\Sigma_{i=1}^{NC} B_i(\xi_l) y_i'(t) = f(t, x, U(\xi_l, t), U(\xi_l, t)_x, U(\xi_l, t)_{xx}),\ l = 1, 2, \cdots, NC,$$

$$b_R(t, U(1,t), U(1,t)_x) = 0.$$

We can determine the coefficients $y_i(t)$ by solving this system of equations using a DAE solver and from there we can get the approximate solution, $U(x,t)$.

3.2 Software for 1D, Time-Dependent, Second-Order PDEs

As mentioned earlier in the thesis, an important feature of some of the most recent software packages for the numerical solution of PDEs from the problem class con-

sidered in this thesis is called adaptive error control. Among all available numerical software for solving 1-dimensional, second-order, time-dependent parabolic PDEs, only HPSIRK [30], HPDASSL [30], HPNEW, BACOL and BACOLR are able to control the overall error, which has two components, the error in space and the error in time.

Assume a general system of NPDE 1-dimensional, second-order, time-dependent PDEs having the general form:

$$\mathbf{u}_t(x,t) = \mathbf{f}(x,t,\mathbf{u},\mathbf{u}_x,\mathbf{u}_{xx}), \qquad a \le x \le b, \qquad t \ge t_0,$$

supplemented with the initial conditions

$$\mathbf{u}(x,t_0) = \mathbf{u}_0(x), \qquad a \le x \le b,$$

and separated boundary conditions

$$\mathbf{b}_L(t,\mathbf{u}(a,t),\mathbf{u}_x(a,t)) = 0, \qquad \mathbf{b}_R(t,\mathbf{u}(b,t),\mathbf{u}_x(b,t)) = 0, \qquad t \ge t_0,$$

where

$$\mathbf{u} = (u_1, u_2, \cdots, u_{NPDE}),$$
$$\mathbf{u}_t = \left(\frac{\partial u_1}{\partial t}, \frac{\partial u_2}{\partial t}, \cdots, \frac{\partial u_{NPDE}}{\partial t}\right),$$
$$\mathbf{u}_x = \left(\frac{\partial u_1}{\partial x}, \frac{\partial u_2}{\partial x}, \cdots, \frac{\partial u_{NPDE}}{\partial x}\right),$$
$$\mathbf{u}_{xx} = \left(\frac{\partial^2 u_1}{\partial x^2}, \frac{\partial^2 u_2}{\partial x^2}, \cdots, \frac{\partial^2 u_{NPDE}}{\partial x^2}\right),$$
$$\mathbf{f} = (f_1, f_2, \cdots, f_{NPDE}).$$

In this section, we review the software packages BACOL and BACOLR and three other software packages, HPNEW, *pdepe* and MOVCOL, all of which are able to solve PDEs having the general form described above.

3.2.1 *BACOL/BACOLR*

A novel feature of BACOL, designed by Wang, Keast, and Muir, is that it employs high order, i.e., high accuracy, adaptive methods in both time and space, controlling and balancing both spatial and temporal error estimates.

For the spatial discretization, BACOL uses B-spline collocation. The resultant DAEs are solved using DASSL, which controls an estimate of the temporal error. For the spatial error estimate, BACOL computes two solutions, one of order p and one of order $p+1$, (p is determined from $KCOL$, the number of collocation points per subinterval, provided by the user) and the difference between the two is used to estimate the error in the lower order solution. BACOL defines a normalized error

estimate for each PDE component over the whole problem interval and a normalized error estimate for each subinterval over all components. As mentioned earlier, after the spatial discretization is performed, BACOL uses DASSL to compute values for the B-spline coefficients, $y_i(t)$. After each successful time step in DASSL, the spatial error is estimated. If the estimated error does not satisfy the tolerance, then BACOL adjusts the mesh and repeats the step. The remeshing strategy involves a procedure for estimating the optimal number of mesh points, and an algorithm for redistributing the mesh points over the problem interval, called an equidistribution algorithm, that tries to make the error on each subinterval the same, and also less than the user provided tolerance.

DASSL uses a Newton iteration to compute the $y_i(t)$ values, and the Newton matrices have a special almost block diagonal (ABD) structure because any B-spline basis function is non-zero only in a small region of the problem domain. BACOL employs COLROW [13] as a linear system solver to take advantage of this special structure. Numerical results [35] show that BACOL is one of the most efficient software packages available whether high accuracy or low accuracy is required. In this thesis work, we employed a slightly modified version of BACOL[1], that replaces COLROW with CONDCOLROW, a package for solving ABD systems that also provides an estimate of the condition number of the ABD matrix. We have introduced this change because it has turned out to be important in this project to be able to monitor the condition numbers of the Newton matrices that arise. One of the modifications

[1]developed by Keast

we need to make to BACOL involves the introduction of new scalings for certain rows of the Newton matrix in an attempt to improve the conditioning of that matrix.

BACOLR was developed by Wang, Keast, and Muir, through a substantial modification of BACOL. For the spatial discretization, BACOL and BACOLR are the same. The novel feature is that the time dependent ODEs resulting from the spatial discretization are handled by a substantially modified version of the Runge-Kutta solver, RADAU5. Numerical results [37] show that the performance of BACOLR is generally comparable to that of BACOL and in some cases significantly superior to that of BACOL.

3.2.2 *HPNEW*

HPNEW, by Moore, a modification of HPDASSL, is another package with spatial and temporal error control. For the spatial discretization, it employs a finite-element Galerkin technique. For the time integration, it uses DASSL. The error estimates are obtained by a formula which generalizes the traditional formula for the Lagrange interpolation error. For the adaptive strategy, HPNEW applies hp-refinement which is a combination of h-refinement and p-refinement. h-refinement involves refining the mesh and p-refinement involves adaptively choosing the degree of the piecewise polynomials used.

3.2.3 *pdepe*

The MATLAB function *pdepe* is a PDE initial-boundary value problem solver. It can solve coupled systems of parabolic and elliptic PDEs in one dimension. There should be at least one parabolic equation in the system. However, it can only control the error in time. *pdepe* solves PDEs of form:

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right) + s\left(x,t,u,\frac{\partial u}{\partial x}\right) \tag{3.1}$$

with the initial condition

$$u(x,t_0) = u_0(x),$$

and the boundary conditions

$$p(x,t,u) + q(x,t)f\left(x,t,u,\frac{\partial u}{\partial x}\right) = 0.$$

where m can be 0, 1, 2; f and s are given functions of $x, t, u, \frac{\partial u}{\partial x}$; $c\left(x,t,u,\frac{\partial u}{\partial x}\right)$ is a diagonal matrix with elements either identically zero or positive; $p(x,t,u)$ is a vector which can depend on u; $q(x,t)$ is a diagonal matrix with elements that are either identically zero or never zero.

3.2.4 *MOVCOL*

MOVCOL, by Huang and Russell, is an example of a moving mesh package. This means that the locations of the spatial mesh points move within the spatial domain as a function of time to adapt to the way the solution changes. Mesh movement is determined through the solution of an auxiliary set of PDEs known as moving mesh PDEs. Central finite differences are used for the spatial discretization for these PDEs. MOVCOL employs a cubic Hermite spline collocation method for the spatial discretization of the physical PDEs. For the time integration, MOVCOL uses DASSL to solve the DAEs. While MOVCOL can adapt the *locations* of the mesh points to the solution behavior, it cannot change the *number* of points and therefore cannot control the spatial error.

3.3 Software for 1D, Time-Dependent, Fourth Order PDEs

A general system of NPDE 1-dimensional, fourth order, time-dependent parabolic PDEs that arises in mathematical models has the form:

$$\mathbf{u}_t(x,t) = \mathbf{f}(x,t,\mathbf{u},\mathbf{u}_x,\mathbf{u}_{xx},\mathbf{u}_{xxx},\mathbf{u}_{xxxx}) \quad a \leq x \leq b, \quad t \geq t_0,$$

supplemented with the initial conditions

$$\mathbf{u}(x, t_0) = \mathbf{u}_0(x), \qquad a \leq x \leq b,$$

and separated boundary conditions

$$\mathbf{b}_L(t, \mathbf{u}(a,t), \mathbf{u}_x(a,t), \mathbf{u}_{xx}(a,t), \mathbf{u}_{xxx}(a,t)) = 0, \qquad t \geq t_0,$$

$$\mathbf{b}_R(t, \mathbf{u}(b,t), \mathbf{u}_x(b,t), \mathbf{u}_{xx}(b,t), \mathbf{u}_{xxx}(b,t)) = 0, \qquad t \geq t_0,$$

where $\mathbf{b}_L \in \Re^q$, $\mathbf{b}_R \in \Re^s$, and $q, s \in \{1, \cdots, 4 \times NPDE\}$,

$$\begin{aligned}
&\mathbf{u} = (u_1, u_2, \cdots, u_{NPDE}), \\
&\mathbf{u}_t = \left(\frac{\partial u_1}{\partial t}, \frac{\partial u_2}{\partial t}, \cdots, \frac{\partial u_{NPDE}}{\partial t} \right), \\
&\mathbf{u}_x = \left(\frac{\partial u_1}{\partial x}, \frac{\partial u_2}{\partial x}, \cdots, \frac{\partial u_{NPDE}}{\partial x} \right), \\
&\mathbf{u}_{xx} = \left(\frac{\partial^2 u_1}{\partial x^2}, \frac{\partial^2 u_2}{\partial x^2}, \cdots, \frac{\partial^2 u_{NPDE}}{\partial x^2} \right), \\
&\mathbf{u}_{xxx} = \left(\frac{\partial^3 u_1}{\partial x^3}, \frac{\partial^3 u_2}{\partial x^3}, \cdots, \frac{\partial^3 u_{NPDE}}{\partial x^3} \right), \\
&\mathbf{u}_{xxxx} = \left(\frac{\partial^4 u_1}{\partial x^4}, \frac{\partial^4 u_2}{\partial x^4}, \cdots, \frac{\partial^4 u_{NPDE}}{\partial x^4} \right), \\
&\mathbf{f} = (f_1, f_2, \cdots, f_{NPDE}) .
\end{aligned}$$

MOVCOL4 is a software package that can solve PDEs which depend on the fourth spatial derivative. MOVCOL4 is based on MOVCOL. For the spatial discretization, MOVCOL4 can employ either collocation or a conservative method. The imple-

mentation of the collocation method is almost the same as in MOVCOL. The small difference is that the degree of the Hermite spline basis in MOVCOL is 3 while in MOVCOL4 it is 7. For the time integration, MOVCOL4 applies the same software, DASSL, that MOVCOL uses. Like MOVCOL, the MOVCOL4 package can control the temporal error but not the spatial error; it is however, to our knowledge, the first software package that is able to solve fourth order PDES.

Chapter 4

Extension of BACOL to Fourth Order Equations

As mentioned previously, for the test problems in [35], BACOL has been shown to be superior to many currently available software packages for solving second order parabolic PDEs. Furthermore BACOL employs adaptive error control in both time and space. However it was not designed to handle fourth order PDEs. The package MOVCOL4, which handles fourth order equations, does not have error control in space. Therefore our research has focused on how to adapt BACOL to handle PDEs in which u_x, u_{xx}, u_{xxx}, and u_{xxxx} appear, so that we can solve fourth order parabolic PDEs with spatial and temporal error control.

We consider a single fourth order PDE (1.1). As mentioned earlier, the authors of the paper [33] point out that converting the fourth order PDE to a system of one second order PDE and one second order ODE(in space) gives a mathematically

equivalent problem. For example, Equation (1.1) can be converted into Equation (1.2). In Section 4.1 we will discuss how to employ the original BACOL to handle the ϵ version of (1.2) - (1.3). In Section 4.2 we will discuss how to adapt and extend BACOL to handle equations of the form (1.2). This new software is called BACOL42.

4.1 BACOL Applied to the ϵ Version of a Problem

During the development of BACOL42 we found that it was useful, for comparison purposes, to be able to compute approximate solutions to (1.2). Since the original BACOL package is able to treat systems of second order PDEs, we observed that a small perturbation of (1.2) could lead to a problem (a) whose solution would hopefully be close to that of (1.2) and (b) that could be treated directly by the original BACOL code. We modified (1.2) to get (1.3).

For sufficiently small ϵ, the perturbed problem (1.3) is obviously close to the converted system (1.2). If ϵ is small enough, then we can therefore expect that the solution of (1.3) is a reasonable approximation to the solution of (1.2).

Thus solving (1.3) with BACOL provides a simple way to obtain an approximation to the solution of a fourth order PDE. Using a version of BACOL modified to provide an estimate of the condition numbers of linear systems that arise, we have seen from our numerical experiments that ϵ cannot be set too small or the Newton iteration matrix will become numerically singular, since we have observed from our numerical experiments that the condition number grows inversely proportional to ϵ.

For each test problem we consider, we will use the original BACOL to solve the

ϵ version of the problem (1.2), i.e., (1.3). The main purpose of this approach is to provide numerical results to compare with the results of the BACOL42.

4.2 Solving the Converted System

4.2.1 Construction of the Newton System

In (1.2), let

$$\underline{f} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} f(x,t,u_1,(u_1)_x,(u_1)_{xx},(u_2)_x,(u_2)_{xx}) \\ (u_1)_{xx} - u_2 \end{bmatrix}$$

and

$$\mathbf{u}(x,t) = \begin{bmatrix} u_1(x,t) \\ u_2(x,t) \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{NC} y_{1,j}(t)B_j(x) \\ \sum_{j=1}^{NC} y_{2,j}(t)B_j(x) \end{bmatrix},$$

where $NC = NINT * KCOL + 4$, $B_j(x)$ are the B-spline basis functions, and $y_{1,j}(t)$ and $y_{2,j}(t)$ are the time-dependent B-spline coefficients of $u_1(x,t)$ and $u_2(x,t)$ respectively. Then:

$$\mathbf{u}_t(x,t) = \sum_{j=1}^{NC} \underline{y}_j'(t)B_j(x)$$
$$\mathbf{u}_x(x,t) = \sum_{j=1}^{NC} \underline{y}_j(t)B_j'(x)$$
$$\mathbf{u}_{xx}(x,t) = \sum_{j=1}^{NC} \underline{y}_j(t)B_j''(x),$$

where $\underline{y}_j(t) = \begin{bmatrix} y_{1,j}(t) \\ y_{2,j}(t) \end{bmatrix}$ and $\underline{y}'_j(t) = \begin{bmatrix} y'_{1,j}(t) \\ y'_{2,j}(t) \end{bmatrix}$.

The converted system with boundary conditions is

$$\begin{cases} 0 = \mathbf{b}_L(t, u(a,t), u_x(a,t), u_{xx}(a,t), u_{xxx}(a,t)) \\ (u_1)_t = f(x, t, u_1, (u_1)_x, (u_1)_{xx}, (u_2)_x, (u_2)_{xx}) \\ 0 = (u_1)_{xx} - u_2 \\ 0 = \mathbf{b}_R(t, u(b,t), u_x(b,t), u_{xx}(b,t), u_{xxx}(b,t)) \end{cases}$$

We first rewrite it by subtracting the right hand side of the system from the left hand side of the system. Collocation then gives a system of ODEs, which together with boundary conditions, gives the DAE system

$$\underline{G}(t, \underline{y}(t), \underline{y}'(t)) = 0,$$

where

$$\underline{y}(t) = \begin{pmatrix} y_{1,1}(t) \\ y_{2,1}(t) \\ y_{1,2,}(t) \\ y_{2,2}(t) \\ \vdots \\ y_{1,NC-1}(t) \\ y_{2,NC-1}(t) \\ y_{1,NC}(t) \\ y_{2,NC}(t) \end{pmatrix}.$$

$\underline{G}$ has three parts:

1. The first 2 components are

$$-b_{L,1}\left(t, a, u_1(a,t), (u_1)_x(a,t), u_2(a,t), (u_2)_x(a,t)\right),$$

$$-b_{L,2}\left(t, a, u_1(a,t), (u_1)_x(a,t), u_2(a,t), (u_2)_x(a,t)\right),$$

or,

$$-b_{L,1}\left(t, a, \sum_{j=1}^{NC} y_{1,j}(t)B_j(a), \sum_{j=1}^{NC} y_{1,j}(t)B_j'(a), \sum_{j=1}^{NC} y_{2,j}(t)B_j(a), \sum_{j=1}^{NC} y_{2,j}(t)B_j'(a)\right),$$

$$-b_{L,2}\left(t, a, \sum_{j=1}^{NC} y_{1,j}(t)B_j(a), \sum_{j=1}^{NC} y_{1,j}(t)B_j'(a), \sum_{j=1}^{NC} y_{2,j}(t)B_j(a), \sum_{j=1}^{NC} y_{2,j}(t)B_j'(a)\right).$$

2. The last 2 components are

$$-b_{R,1}\left(t, b, u_1(b,t), (u_1)_x(b,t), u_2(b,t), (u_2)_x(b,t)\right),$$

$$-b_{R,2}\left(t, b, u_1(b,t), (u_1)_x(b,t), u_2(b,t), (u_2)_x(b,t)\right),$$

or,

$$-b_{R,1}\left(t, b, \sum_{j=1}^{NC} y_{1,j}(t)B_j(b), \sum_{j=1}^{NC} y_{1,j}(t)B_j'(b), \sum_{j=1}^{NC} y_{2,j}(t)B_j(b), \sum_{j=1}^{NC} y_{2,j}(t)B_j'(b)\right),$$

$$-b_{R,2}\left(t, b, \sum_{j=1}^{NC} y_{1,j}(t)B_j(b), \sum_{j=1}^{NC} y_{1,j}(t)B_j'(b), \sum_{j=1}^{NC} y_{2,j}(t)B_j(b), \sum_{j=1}^{NC} y_{2,j}(t)B_j'(b)\right).$$

3. The remaining middle components are of the form (these are the collocation equations)

$$(u_1)_t(\xi_l, t) - f\left(t, \xi_l, u_1(\xi_l,t), (u_1)_x(\xi_l,t), (u_2)_x(\xi_l,t), (u_1)_{xx}(\xi_l,t), (u_2)_{xx}(\xi_l,t)\right),$$

$$u_2(\xi_l, t) - (u_1)_{xx}(\xi_l, t),$$

or,

$$\begin{aligned}
\sum_{j=1}^{NC} \underline{y}_j'(t) B_j(\xi_l) \quad &- \quad f(\xi_l, t, \sum_i^{NC} \underline{y}_{1,i}(t) B_i(\xi_l), \sum_i^{NC} \underline{y}_{1,i}(t) B_i'(\xi_l), \\
&\qquad \sum_i^{NC} \underline{y}_{2,i}(t) B_i'(\xi_l), \sum_i^{NC} \underline{y}_{1,i}(t) B_i''(\xi_l), \sum_i^{NC} \underline{y}_{2,i}(t) B_i''(\xi_l)), \\
\sum_{j=1}^{NC} \underline{y}_{2,j}(t) B_j(x) \quad &- \quad \sum_{j=1}^{NC} \underline{y}_{1,j}(t) B_j''(x),
\end{aligned}$$

where ξ_l is one of the $(p-1)$ collocation points associated with the ith subinterval (p is the degree of the piecewise polynomials). There are $NINT \times (p-1)$ pairs of such equations, where $NINT$ is the number of subintervals.

We can rewrite the DAE system as

$$\underline{G}(t, \underline{y}(t), \underline{y}'(t)) = A\underline{y}'(t) - \underline{\tilde{F}}(t, \underline{y}(t)) = 0. \tag{4.1}$$

Here the matrix A represents the terms of the collocation equations that depend on $\underline{y}'(t)$. The top and bottom parts of $\underline{\tilde{F}}$ correspond to the boundary conditions and the middle part of $\underline{\tilde{F}}$ corresponds to the terms of the collocation equations that do not depend on $\underline{y}'(t)$.

Application of a BDF as the time-stepping formula for the treatment of the above DAE system (4.1) gives a nonlinear system for which the corresponding Newton iteration is

$$PD \cdot \triangle \underline{y}_{n+1}^{(m)} = -\left[A\left(\frac{\alpha}{h_{n+1}} \underline{y}_{n+1}^{(m)} + \beta \right) - \underline{\tilde{F}}(t_{n+1}, \underline{y}_{n+1}^{(m)}) \right], \tag{4.2}$$

where $\underline{y}_{n+1}^{(m+1)} = \underline{y}_{n+1}^{(m)} + \triangle \underline{y}_{n+1}^{(m)}$, α is the coefficient of $\underline{y}_{n+1}$ in the BDF, h_{n+1} is the current timestep, and β represents the part of the BDF that depends on known solution information from previous timesteps. PD, the Newton matrix, is

$$PD = c_j \frac{\partial \underline{G}}{\partial \underline{y'}} + \frac{\partial \underline{G}}{\partial \underline{y}} = c_j \frac{\partial G}{\partial \underline{y'}} - \frac{\partial \underline{\tilde{F}}}{\partial \underline{y}} = c_j A - \frac{\partial \underline{\tilde{F}}}{\partial \underline{y}},$$

where $c_j = \frac{\alpha}{h_{n+1}}$.

The rows of the matrix A corresponding to the collocation equations that depend on components of $\underline{y}'(t)$ have the form:

$$\begin{pmatrix} B_1(\xi_l)I & B_2(\xi_l)I & \cdots & B_{NC}(\xi_l)I \end{pmatrix}$$

where $B_j(\xi_l)I$ is a 2 by 2 diagonal matrix associated with the jth B-spline basis polynomial evaluated at the lth collocation point. The remainder of A has the following structure:

1. The elements of the first two rows and the last two rows are zero (corresponding to the left boundary and right boundary conditions);

2. Since the B-spline basis functions have small compact support, the A matrix has an ABD structure, as mentioned earlier. Every second row of each block in

the middle part of matrix A is filled with zeros, i.e., each block has the form:

$$\begin{pmatrix} x & 0 & x & 0 & \cdots & x & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x & 0 & x & 0 & \cdots & x & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix},$$

where the elements labeled x are usually non-zero values of the B-spline basis functions, and the zero rows correspond to collocating to the second equation (the ODE) of (1.2).

One of the main modifications that needed to be made to BACOL was associated with handling the presence of these zero rows in the A matrix.

The middle part of $\underline{\tilde{F}}(t, y)$ has $(NC - 2) \times 2$ components; the $(2l + 1)$th and $(2l + 2)$th vector components (where $l = 1, 2, \cdots, NC - 2$) are the right hand side of the lth collocation equations,

$$\underline{f}\left(\xi_l, t, \underline{u}(\xi_l, t), \underline{u}_x(\xi_l, t), \underline{u}_{xx}(\xi_l, t)\right),$$

or

$$\underline{f}\left(\xi_l, t, \sum_i^{NC} \underline{y}_i(t)B_i(\xi_l), \sum_i^{NC} \underline{y}_i(t)B_i'(\xi_l), \sum_i^{NC} \underline{y}_i(t)B_i''(\xi_l)\right).$$

(Note that here we are referring to the vector $\underline{f}$ that includes both the PDE and the ODE of the converted system.)

Similarly the boundary condition components of $\underline{\tilde{F}}(t, y)$ have the form

$$\underline{b}_L\left(t, \sum_i^{NC} \underline{y}_i(t)B_i(a), \sum_i^{NC} \underline{y}_i(t)B_i'(a)\right),$$

and

$$\underline{b}_R\left(t, \sum_i^{NC} \underline{y}_i(t)B_i(b), \sum_i^{NC} \underline{y}_i(t)B_i'(b)\right).$$

The components of the matrix $\frac{\partial \underline{\tilde{F}}}{\partial \underline{y}}$ are obtained by differentiating the components of $\underline{\tilde{F}}$ with respect to the components of $\underline{y}$. Therefore the top part of the Newton system (4.2) corresponding to the left boundary condition has the form:

$$-\frac{\partial \underline{b}_L}{\partial \underline{y}}\left(\triangle \underline{y}_{n+1}^{(m)}\right) = -\underline{b}_L,$$

where

$$-\frac{\partial \underline{b}_L}{\partial \underline{y}} = -\left[\frac{\partial}{\partial \underline{y}_1}\underline{b}_L \quad \frac{\partial}{\partial \underline{y}_2}\underline{b}_L \quad \cdots \quad \frac{\partial}{\partial \underline{y}_{NC}}\underline{b}_L\right],$$

where

$$\frac{\partial}{\partial \underline{y}_j}\underline{b}_L = \frac{\partial}{\partial \underline{y}_j}\underline{b}_L\left(t, \sum_{i=1}^{NC} \underline{y}_i(t)B_i(a), \sum_{i=1}^{NC} \underline{y}_i(t)B_i'(a)\right)$$

becomes

$$\frac{\partial \underline{b}_L}{\partial \underline{u}} B_j(a) + \frac{\partial \underline{b}_L}{\partial \underline{u}_x} B_j'(a), \qquad (a \;\; 2 \times 2 \;\; matrix).$$

Since the B-splines have small compact support most of the top row of $\frac{\partial \underline{\tilde{F}}}{\partial \underline{y}}$ consists of zero blocks except for the first few positions and most of the last block row of $\frac{\partial \underline{\tilde{F}}}{\partial \underline{y}}$ is zero except for the last few positions. The bottom part corresponding to the right boundary condition is similar to the top part, shown above.

The remaining rows of $\frac{\partial \underline{\tilde{F}}}{\partial \underline{y}}$ are obtained by considering the derivative of the right hand side of the lth pair of collocation equations with respect to $\underline{y}$. The lth block row of $\frac{\partial \underline{\tilde{F}}}{\partial \underline{y}}$ is

$$\left[\frac{\partial}{\partial \underline{y}_1} \underline{f} \;\; \frac{\partial}{\partial \underline{y}_2} \underline{f} \;\; \cdots \;\; \frac{\partial}{\partial \underline{y}_{NC}} \underline{f} \right]$$

and, in detail, the jth component of the above block row is

$$\begin{aligned} & \frac{\partial}{\partial \underline{y}_j} \underline{f} \left(\xi_l, t, \sum_{i=1}^{NC} \underline{y}_i(t) B_i(\xi_l), \sum_{i=1}^{NC} \underline{y}_i(t) B_i'(\xi_l), \sum_{i=1}^{NC} \underline{y}_i(t) B_i''(\xi_l) \right) \\ = \quad & \frac{\partial \underline{f}}{\partial \underline{u}} B_j(\xi_l) + \frac{\partial \underline{f}}{\partial \underline{u}_x} B_j'(\xi_l) + \frac{\partial \underline{f}}{\partial \underline{u}_{xx}} B_j''(\xi_l), \qquad (2 \times 2 \;\; matrix). \end{aligned}$$

Again since the B-splines have small compact support most of the entries in the lth block row are zero except for those corresponding to B-spline functions that are non-zero near the collocation point ξ_l.

4.2.2 Scaling of the Newton System

When we try to solve the converted system, the Newton matrix, PD, turns out to be poorly conditioned. (We have observed this from our numerical experiments in which the condition number of the Newton matrix is reported.) We can see that in the DAE system (4.1) there are 4 algebraic constraints coming from boundary conditions together with $NINT \times (k-2)$ algebraic constraints corresponding to collocation of the second equation in (1.2). From the paper [36], we know that if we do not scale the equations of the DAE system that correspond to the boundary conditions, i.e., if we do not scale the the algebraic equations of the DAE system, large condition numbers arise and BACOL fails due to the ill-conditioning of the corresponding Newton matrix. See [36] and references within for further discussion of this issue.

Therefore our modified version of BACOL employs the technique discussed in [32] of scaling the Jacobian subblocks and right hand side elements corresponding to the algebraic constraints from the ODE and the boundary conditions by c_j. (Previously we defined as $c_j = \frac{\alpha}{h_{time}}$). Consider the top part of the Newton system corresponding to the left boundary condition as an example; recall that it has the form

$$-\frac{\partial \underline{b}_L}{\partial \underline{y}} \left(\triangle \underline{y}_{n+1}^{(m)} \right) = -\underline{b}_L.$$

We will replace this part of the Newton system by the scaled form

$$-c_j \frac{\partial \underline{b}_L}{\partial \underline{y}} \left(\triangle \underline{y}_{n+1}^{(m)} \right) = -c_j \underline{b}_L.$$

The bottom part of the Newton system is treated in the same way; we get:

$$-c_j \frac{\partial \underline{b}_R}{\partial \underline{y}} \left(\triangle \underline{y}_{n+1}^{(m)} \right) = -c_j \underline{b}_R.$$

Every second component of the middle part of the Newton system corresponding to algebraic constraints from the ODE should be scaled by c_j as well.

The general form of the Newton system is:

$$\left(c_j A - \frac{\partial \underline{\tilde{F}}}{\partial \underline{y}} \right) \triangle \underline{y}_{n+1}^{(m)} = - \left[A \left(\frac{\alpha}{h_{n+1}} \underline{y}_{n+1}^{(m)} + \beta \right) - \underline{\tilde{F}}(t_{n+1}, \underline{y}_{n+1}^{(m)}) \right]. \tag{4.3}$$

As we know, every second row of the middle part of A is zero. Therefore the $(2l+2)$th row corresponding to the lth collocation point ξ_l is

$$-\frac{\partial f_2(\xi_l)}{\partial \underline{y}} \left(\triangle \underline{y}_{n+1}^{(m)} \right) = -f_2(\xi_l)$$

becomes

$$-c_j \frac{\partial f_2(\xi_l)}{\partial \underline{y}} \left(\triangle \underline{y}_{n+1}^{(m)} \right) = -c_j f_2(\xi_l). \tag{4.4}$$

4.3 Summary of Modifications to BACOL

From the discussion in the previous section, the efficient treatment of the converted system, (1.2), by BACOL requires that we handle two issues: (i) we have to deal with the fact that the equations arising from the application of collocation to the ODE

components of the converted system do not depend on the time derivative of the solution to the DAE system (and this means that every second row of the A matrix arising in (4.3) should consist of zero entries), and (ii) because of the absence of the time derivative in these equations, within the DAE system they represent algebraic equations, and it is well-known that the algebraic equations in a DAE system to be treated by DASSL must be scaled appropriately (this scaling was already present in BACOL for the boundary conditions).

In order to modify BACOL to deal with these issues, an extensive study of the source code of the BACOL package was required. At the end of this analysis, it was found that a small but subtle set of modifications could be introduced to handle the above issues. An important subtlety, that became apparent only after much careful investigation, is that the matrix A, actually plays two roles within BACOL. In addition to contributing to the Newton system - see (4.3) - it also plays the role of a projection matrix within the routine which performs an important initialization calculation at the beginning of the computation and an important reinitialization calculation after each remeshing performed by BACOL. The initialization involves projecting the initial solution onto the B-spline basis space to obtain the B-spline coefficients for the solution. The reinitialization involves projecting the current solution onto the B-spline basis associated with the new mesh so that the B-spline coefficients of the current solution can be obtained. *An essential observation is that even for the converted system, the use of the matrix A in the initialization and reinitialization computations must be the same as it is in BACOL, whereas in the computation of the*

Newton system (4.3), the A matrix arising there must have zero rows in certain locations, as explained in the previous subsection. Based on this observation, we realized that the A matrix should not be altered but rather the contribution to the Newton system from the A matrix needed to be modified so that the effect would be the same as if the A matrix did in fact have zero rows.

Based on the analysis from the previous section and our careful investigation of the BACOL source code, we found there are two types of modifications involving three subroutines in the BACOL source code which needed to be changed to handle the second order mixed PDE/ODE system (1.2).

The first modification is associated with the extra zero rows of the matrix A corresponding to the ODE. The purpose of this change is to handle calculations associated with the zeros in the matrix A. This modification involves two subroutines: *CALJAC* and *CALRES*, which interface with DASSL to handle the treatment of the Newton systems associated with the discretization of the converted PDE/ODE system.

The second type of modification is associated with scaling the additional algebraic constraints arising from collocating the ODE part of the converted system. Two subroutines are required to be changed: *CALJAC* which scales the left hand side of the equation (4.4) and *DDASLV* which handles the scaling of the right hand side of the equation (4.4).

We give detailed descriptions of the modifications in the Appendix.

Chapter 5

Numerical Results for BACOL42

In this chapter we present numerical results to explore the effectiveness of the modified version of BACOL, called BACOL42, that we described in the previous chapter. For comparison purposes we also provide results obtained by applying the original BACOL to the ϵ version of each problem.

We will consider five different time-dependent 1D fourth order PDEs. We use GNU Fortran77 (GCC) 3.4.6 under the linux operation system (ubuntu 8.04) running on an HP 380DL G5 (processor speed 2.33 GHz). Section 5.1 discusses the simple test equation (2.12). In Section 5.2, we consider the numerical results for equation (2.13). The problem in Section 5.3 comes from thin film liquids (5.5). In Section 5.4, we apply the software to the Kuramoto-Sivashinsky Equation (2.7). Finally, in Section 5.5, we consider the Cahn-Hilliard Equation (5.10). For each of these problems, we first convert the fourth order PDE to a system of second order equations; that is, one second order PDE and one second order ODE as in (1.2); we also consider the ϵ

version of each converted system as in (1.3). Second, we use BACOL42 and BACOL to solve the problem and plot the solution using MATLAB. Finally, we provide tables displaying the performance of both BACOL42 and BACOL for different values of $KCOL$ and tolerance. In addition, for various tolerances and $KCOL$ values, we give tables showing the L^2-norm errors for both BACOL42 and BACOL for the first two problems, where we know the exact solution. The L^2-norm errors are defined as

$$\sqrt{\frac{\sum_{i=1}^{NINT+1}(u_{exact}(x_i,t)-u_{approx}(x_i,t))^2}{NINT+1}},$$

where $u_{exact}(x,t)$ is the exact solution at time t and $u_{approx}(x,t)$ is the approximation solution at time t, and x_i is the ith mesh point. At the end of this chapter we discuss the results to compare the performances of BACOL42 and BACOL.

Note:

In each section, we give a table showing how BACOL42 or BACOL (applied to the ϵ version of the converted system) performs. In these tables we use the following symbols.

- 1: BACOL42 or BACOL returns with the DASSL error condition 8: the iteration matrix is numerically singular;
- 2: BACOL42 or BACOL returns with the error message that the code has remeshed 20 times at $t = 0$ and failed to start;
- 3: BACOL42 or BACOL returns with the DASSL error condition 7: the corrector failed to converge repeatedly or the stepsize has been reduced to a set minimun, $HMIN$;
- ✓: BACOL42 or BACOL returns and reports that it has solved the problem to within the requested tolerance;
- −: BACOL42 or BACOL is unable to make a successful start at $t = 0$ but does not return with an error message as in 2 above. Rather the code continues to compute (for many hours) without making any progress. It appears that DASSL is making slow but sufficient progress that it does not encounter an error condition; however the computation proceeds very slowly despite not triggering an error condition. A closer examination of the detailed computations being performed would have to be undertaken to better understand what is happening in this case.

Also, *atol* is the absolute tolerance, *rtol* is the relative tolerance, and computations were done in double precision.

5.1 Simple Test Problem One

The first test problem, (2.12)), is the simple Fourth order linear PDE discussed in Section 2.7.1. The exact solution (see Figure 2.4) is

$$u(x,t) = e^{-t}\sin x.$$

Letting $u_1(x,t) = u(x,t)$ and $u_2(x,t) = (u_1)_{xx}(x,t)$, we convert the above equation to the following system:

$$\begin{cases} (u_1)_t = -(u_2)_{xx}, & 0 \le x \le \pi, \quad t > 0, \\ 0 = (u_1)_{xx} - u_2, & 0 \le x \le \pi, \quad t > 0, \\ u_1(0,t) = u_1(\pi,t) = 0, & t > 0, \\ u_2(0,t) = u_2(\pi,t) = 0, & t > 0, \\ u_1(x,0) = \sin x, & 0 \le x \le \pi, \\ u_2(x,0) = -\sin x, & 0 \le x \le \pi. \end{cases} \tag{5.1}$$

If we let $KCOL = 2$ and $atol = rtol = 10^{-6}$, we can obtain the approximate solution in Figure 5.1 for $0 \le x \le \pi$ and $0 \le t \le 1$.

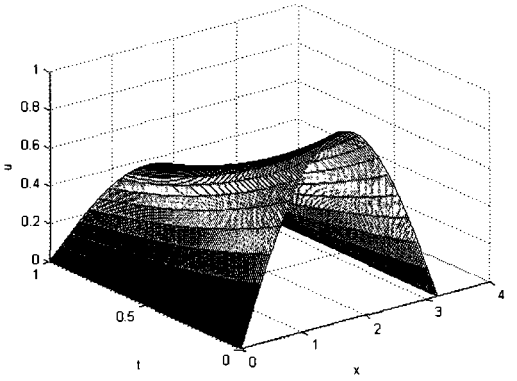

Figure 5.1: Approximate Solution of Example One from BACOL42

BACOL42 can solve this equation for $KCOL$ in the range from 2 to 10 (the maximum allowed in BACOL42) and for tolerances from 10^{-3} to 10^{-7}. When $atol = rtol = 10^{-8}$, the condition number of the Newton iteration matrix becomes too large (as large as 10^{22}) and BACOL42 reports that the iteration matrix is numerically singular. Further investigation would be required in order to better understand this issue. Sharper tolerances lead to meshes with many more subintervals which in turn leads to larger matrices; it may be the case that the condition numbers of these matrices grow with the number of subintervals. Table 5.1 displays how BACOL42 works with the different values of $KCOL$ and tolerance for Example One. From Table 5.2 we can see the relationship between the tolerance and the global error for BACOL for Example One.

Table 5.1: Performance of BACOL42 with different $KCOL$ and tolerance values for Example One

TOL	KCOL								
	2	3	4	5	6	7	8	9	10
10^{-4}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-5}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-6}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-7}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-8}	1	1	1	1	1	1	1	1	1
10^{-9}	1	1	1	1	1	1	1	1	1
10^{-10}	1	1	1	1	1	1	1	1	1

Table 5.2: L^2-Error Norms from BACOL42 for Example One

TOL	KCOL				
	2	4	6	8	10
10^{-4}	1.56E-04	1.58E-04	1.64E-04	1.57E-04	1.57E-04
10^{-5}	1.60E-05	1.34E-05	1.40E-05	1.60E-05	1.60E-05
10^{-6}	8.97E-07	1.03E-06	3.40E-06	1.42E-06	1.47E-06
10^{-7}	1.43E-07	1.55E-07	1.23E-07	3.55E-07	1.83E-07

In order to apply the original BACOL, we consider the approximate form of the system (5.1)

$$\begin{cases} (u_1)_t = -(u_2)_{xx}, & 0 \le x \le \pi, \quad t > 0, \\ \epsilon(u_2)_t = (u_1)_{xx} - u_2, & 0 \le x \le \pi, \quad t > 0, \\ u_1(0,t) = u_1(\pi,t) = 0, & t > 0, \\ u_2(0,t) = u_2(\pi,t) = 0, & t > 0, \\ u_1(x,0) = \sin x, & 0 \le x \le \pi, \\ u_2(x,0) = -\sin x, & 0 \le x \le \pi. \end{cases} \tag{5.2}$$

We will choose ϵ to have the same value as the tolerance for every case. (It is not clear how to choose ϵ. It would appear that it should be chosen to be less than or equal to the tolerance but without knowing the exact relationship between the value of ϵ and the difference between the solution to the original converted system and the ϵ approximation to the converted system, it is hard to know how to choose ϵ.) BACOL can solve (5.2) for $KCOL$ in the range from 2 to 10 and for tolerances from 10^{-3} to 10^{-6}. Table 5.3 displays how BACOL works with the different values of $KCOL$ and tolerance for Example One. Table 5.4 shows the relationship between the tolerance and the global error for BACOL for Example One.

Table 5.3: Performance of BACOL with different $KCOL$ and tolerance values for Example One

TOL	KCOL								
	2	3	4	5	6	7	8	9	10
10^{-4}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-5}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-6}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-7}	-	-	-	-	-	-	-	-	-
10^{-8}	-	-	-	-	-	-	-	-	-
10^{-9}	-	-	-	-	-	-	-	-	-
10^{-10}	-	-	-	-	-	-	-	-	-

Table 5.4: L^2-Error Norms from BACOL for Example One

TOL	KCOL				
	2	4	6	8	10
10^{-4}	1.79E-04	1.75E-04	1.84E-04	1.84E-04	1.80E-04
10^{-5}	9.59E-06	1.27E-05	1.47E-05	1.46E-05	1.40E-05
10^{-6}	3.51E-06	3.86E-06	4.43E-06	2.03E-06	2.92E-06

5.2 Simple Test Problem Two

The second example, (2.13), was introduced in Section 2.7.2. When $0 \leq x \leq \pi$ and $0 < t < 0.6$, the exact solution (see Figure 2.5) is

$$u(x,t) = 0.3(\cos(t) + 3)\cos x.$$

Here t is restricted to be less than π. If this restriction is not met, then the coefficient, $-\frac{\sin t}{\cos t+3}$, will be positive (Recall that the PDE has the form $u_t = -\left(\frac{\sin t}{3+\cos t}\right) u_{xxxx}$). Then the PDE will become unstable and thus difficult for any numerical software package to solve.

We convert equation (2.13) into the following system:

$$\begin{cases} (u_1)_t = -\frac{\sin t}{\cos t+3}(u_2)_{xx}, & 0 \leq x \leq \pi, \quad t > 0, \\ 0 = (u_1)_{xx} - (u_2), & 0 \leq x \leq \pi, \quad t > 0, \\ u_1(x,0) = 1.2\cos(x), & 0 \leq x \leq \pi, \\ u_2(x,0) = -1.2\cos(x), & 0 \leq x \leq \pi, \\ u_1(0,t) = u_2(\pi,t) = 0.3(\cos(t) + 3), & t > 0, \\ u_1(\pi,t) = u_2(0,t) = -0.3(\cos(t) + 3), & t > 0. \end{cases} \tag{5.3}$$

When $KCOL = 2$ and $atol = rtol = 10^{-6}$, the approximate solution ($0 \leq x \leq \pi$ and $0 < t < 0.6$) is plotted in Figure 5.2.

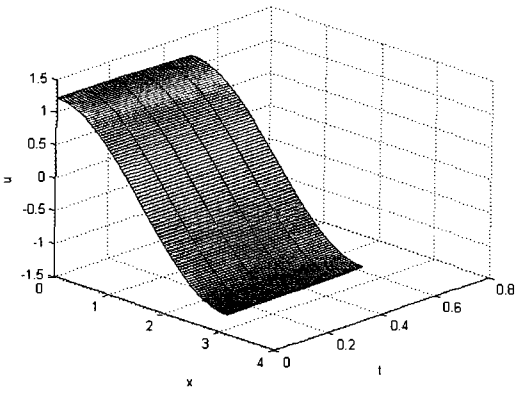

Figure 5.2: Approximate Solution of Example Two from BACOL42

Here we also provide Table 5.5 that displays how BACOL42 works with the different values of $KCOL$ and tolerance for Example Two. We can see BACOL42 can solve this equation for $KCOL$ in the range from 2 to 10 and for tolerances from 10^{-4} to 10^{-9} (in some cases). From Table 5.6 we can see the relationship between the tolerance and the global error for BACOL42 for Example Two. ("NA" means that BACOL42 reports an error message and no solution is available.)

Table 5.5: Performance of BACOL42 with different $KCOL$ and tolerance values for Example Two

TOL	KCOL								
	2	3	4	5	6	7	8	9	10
10^{-5}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-6}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-7}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-8}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-9}	1	✓	✓	✓	✓	3	3	3	3
10^{-10}	1	1	1	1	1	1	3	3	1

Table 5.6: L^2-Error Norms from BACOL42 for Example Two

TOL	KCOL				
	2	4	6	8	10
10^{-5}	1.98E-06	2.14E-06	1.38E-06	1.43E-06	2.33E-06
10^{-6}	8.53E-07	7.83E-07	7.28E-07	7.41E-07	8.55E-07
10^{-7}	2.29E-07	1.96E-07	1.80E-07	2.06E-07	2.12E-07
10^{-8}	4.82E-08	4.82E-08	4.08E-08	4.68E-08	4.95E-08
10^{-9}	NA	1.15E-08	9.63E-09	NA	NA

In order to apply BACOL, we consider the following approximate form of the system (5.3)

$$\begin{cases} (u_1)_t = -\frac{\sin t}{\cos t+3}(u_2)_{xx}, & 0 \le x \le \pi, \quad t > 0, \\ \epsilon(u_2)_t = (u_1)_{xx} - (u_2), & 0 \le x \le \pi, \quad t > 0, \\ u_1(x,0) = 1.2\cos(x), & 0 \le x \le \pi, \\ u_2(x,0) = -1.2\cos(x), & 0 \le x \le \pi, \\ u_1(0,t) = u_2(\pi,t) = 0.3(\cos(t)+3), & t > 0, \\ u_1(\pi,t) = u_2(0,t) = -0.3(\cos(t)+3), & t > 0. \end{cases} \tag{5.4}$$

We let ϵ be the same value as the tolerance for every case. BACOL can solve (5.4) for $KCOL$ in the range from 2 to 10 and for tolerances from 10^{-3} to 10^{-9}. Table 5.7 displays how BACOL works with the different values of $KCOL$ and tolerance for Example Two. Table 5.8 displays the relationship between the tolerance and the global error for BACOL for Example Two.

Table 5.7: Performance of BACOL with different $KCOL$ and tolerance values for Example Two

TOL	KCOL								
	2	3	4	5	6	7	8	9	10
10^{-5}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-6}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-7}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-8}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-9}	1	✓	✓	✓	✓	✓	✓	✓	✓
10^{-10}	1	-	-	-	-	-	-	-	-

Table 5.8: L^2-Error Norms from BACOL for Example Two

TOL	KCOL				
	2	4	6	8	10
10^{-5}	4.21E-05	4.90E-05	3.18E-05	3.13E-05	5.38E-05
10^{-6}	2.03E-05	1.85E-05	1.71E-05	1.74E-05	2.01E-05
10^{-7}	5.26E-06	4.81E-06	4.23E-06	4.85E-06	4.98E-06
10^{-8}	1.33E-06	1.18E-06	9.90E-07	1.10E-06	1.06E-06
10^{-9}	NA	2.96E-07	2.15E-07	2.31E-07	2.41E-07

5.3 Thin Film Equation

This problem [3] comes from modeling thin film liquids. We discussed different types of thin-film equations in Section 2.3. The equation we selected, (2.6), has the following form,

$$u_t = -(u^{\frac{1}{2}} u_{xxx})_x, \qquad -1 \leq x \leq 1, \qquad t > 0, \tag{5.5}$$

with the initial condition

$$u(x,0) = 0.8 - \cos(\pi x) + 0.25\cos(2\pi x), \qquad -1 \leq x \leq 1,$$

and the boundary conditions

$$u_x(-1,t) = u_x(1,t) = 0, \qquad t > 0,$$

$$u_{xxx}(-1,t) = u_{xxx}(1,t) = 0, \qquad t > 0.$$

The exact solution is not known for this equation. However, from [33] we know that when t approaches 0.0007302 the solution ceases to exist.

We convert Equation (5.5) to the following system:

$$\begin{cases} (u_1)_t = -((u_1)^{\frac{1}{2}}(u_2)_x)_x, & -1 \le x \le 1, \quad t > 0, \\ 0 = (u_1)_{xx} - u_2, & -1 \le x \le 1, \quad t > 0, \\ (u_1)_x(-1,t) = (u_1)_x(1,t) = 0, & t > 0, \\ (u_2)_x(-1,t) = (u_2)_x(1,t) = 0, & t > 0, \\ u_1(x,0) = 0.8 - \cos(\pi x) + 0.25\cos(2\pi x), & -1 \le x \le 1, \\ u_2(x,0) = \pi^2\cos(\pi x) - \pi^2\cos(2\pi x), & -1 \le x \le 1. \end{cases} \tag{5.6}$$

With $KCOL = 4$ and $atol = rtol = 10^{-7}$, BACOL42 was used to solve the equation. The approximate solution is plotted in Figure 5.3 when $0 \le t \le 0.00073$ and $0 \le x \le 1$.

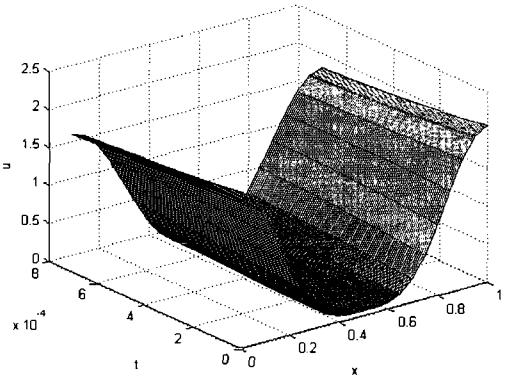

Figure 5.3: Approximate Solution of Example Three from BACOL42

Table 5.9 displays how BACOL42 works with the different values of $KCOL$ and tolerance for Example Three.

Table 5.9: Performance of BACOL42 with different $KCOL$ and tolerance values for Example Three

TOL	KCOL								
	2	3	4	5	6	7	8	9	10
10^{-5}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-6}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-7}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-8}	1	✓	✓	✓	✓	✓	✓	✓	✓
10^{-9}	3	3	✓	✓	✓	✓	✓	✓	✓
10^{-10}	3	3	3	3	✓	✓	✓	✓	✓

Application of BACOL requires the ϵ version of (5.6)

$$
\begin{cases}
(u_1)_t = -((u_1)^{\frac{1}{2}}(u_2)_x)_x, & -1 \le x \le 1, \quad t > 0, \\
\epsilon(u_2)_t = (u_1)_{xx} - u_2, & -1 \le x \le 1, \quad t > 0, \\
(u_1)_x(-1,t) = (u_1)_x(1,t) = 0, & t > 0, \\
(u_2)_x(-1,t) = (u_2)_x(1,t) = 0, & t > 0, \\
u_1(x,0) = 0.8 - \cos(\pi x) + 0.25\cos(2\pi x), & -1 \le x \le 1, \\
u_2(x,0) = \pi^2\cos(\pi x) - \pi^2\cos(2\pi x), & -1 \le x \le 1.
\end{cases}
\tag{5.7}
$$

As before, we choose $\epsilon = TOL$. Table 5.10 displays how BACOL works with the different values of $KCOL$ and tolerance for Example Three. It is interesting that BACOL can solve this problem with $KCOL$ from 2 to 10 and tolerance from 10^{-5} to 10^{-10}.

Table 5.10: Performance of BACOL with different $KCOL$ and tolerance values for Example Three

TOL	KCOL								
	2	3	4	5	6	7	8	9	10
10^{-5}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-6}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-7}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-8}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-9}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-10}	✓	✓	✓	✓	✓	✓	✓	✓	✓

5.4 Kuramoto-Sivashinsky Equation

In section 2.3.1, we introduced the Kuramoto-Sivashinsky Equation (2.7). Recall this equation has the form:

$$u_t = -uu_x - u_{xx} - u_{xxxx}, \quad x \in [0, 32\pi.]$$

The initial condition is

$$u(x,0) = \cos\left(\frac{x}{16}\right)\left(1+\sin\left(\frac{x}{16}\right)\right).$$

The boundary conditions are

$$\begin{aligned} u(0,t) &= u(32\pi,t) = 1, \qquad t>0, \\ u_{xx}(0,t) &= u_{xx}(32\pi,t) = -\frac{1}{16^2}, \qquad t>0. \end{aligned}$$

We convert this fourth order problem to the following system:

$$\begin{cases} (u_1)_t = -u_1(u_1)_x - (u_1)_{xx} - (u_2)_{xx}, \qquad 0 \le x \le 32\pi, \qquad t>0, \\ 0 = (u_1)_{xx} - u_2, \qquad 0 \le x \le 32\pi, \qquad t>0, \\ u_1(0,t) = (u_1)(32\pi,t) = 1, \qquad t>0, \\ u_2(0,t) = (u_2)(32\pi,t) = -\frac{1}{16^2}, \qquad t>0, \\ u_1(x,0) = \cos(\frac{x}{16})(1+\sin(\frac{x}{16})), \qquad 0 \le x \le 32\pi, \\ u_2(x,0) = -\frac{1}{16^2}(\cos(\frac{x}{16}) + 2\sin(\frac{x}{8})), \qquad 0 \le x \le 32\pi. \end{cases} \tag{5.8}$$

Using $KCOL = 2$ and $atol = rtol = 10^{-6}$, when $0 \le t \le 1$ and $0 \le x \le 32\pi$, the approximate solution is plotted in Figure 5.4.

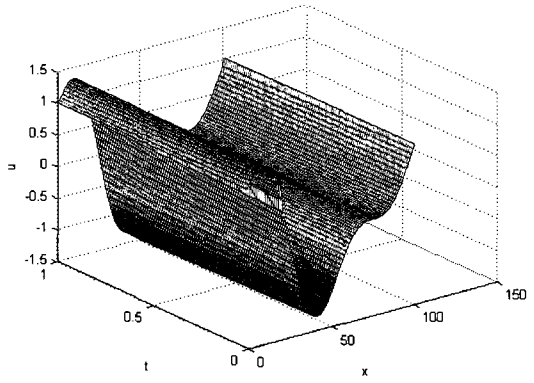

Figure 5.4: Approximate Solution of Example Four from BACOL42

We also considered different $KCOL$ values (from 2 to 10) and tolerance values (from 10^{-5} to 10^{-10}). Table 5.11 displays how BACOL42 works with the different values of $KCOL$ and tolerance for Example Four.

Table 5.11: Performance of BACOL42 with different $KCOL$ and tolerance values for Example Four

TOL	KCOL								
	2	3	4	5	6	7	8	9	10
10^{-5}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-6}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-7}	✓	✓	✓	✓	✓	✓	✓	✓	-
10^{-8}	1	1	1	1	1	1	1	1	1
10^{-9}	1	1	1	1	1	1	1	1	1
10^{-10}	1	1	1	1	1	1	1	1	1

We then applied BACOL to the ϵ version of (5.8)

$$\begin{cases} (u_1)_t = -u_1(u_1)_x - (u_1)_{xx} - (u_2)_{xx}, & 0 \le x \le 32\pi, \quad t > 0, \\ \epsilon(u_2)_t = (u_1)_{xx} - u_2, & 0 \le x \le 32\pi, \quad t > 0, \\ u_1(0,t) = (u_1)(32\pi,t) = 1, & t > 0, \\ u_2(0,t) = (u_2)(32\pi,t) = -\frac{1}{16^2}, & t > 0, \\ u_1(x,0) = \cos(\frac{x}{16})(1+\sin(\frac{x}{16})), & 0 \le x \le 32\pi, \\ u_2(x,0) = -\frac{1}{16^2}(\cos(\frac{x}{16}) + 2\sin(\frac{x}{8})), & 0 \le x \le 32\pi. \end{cases} \tag{5.9}$$

When $\epsilon = TOL$, Table 5.12 displays how BACOL works with the different values of $KCOL$ and tolerance for Example Four.

Table 5.12: Performance of BACOL with different $KCOL$ and tolerance values for Example Four

TOL	KCOL								
	2	3	4	5	6	7	8	9	10
10^{-5}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-6}	-	-	-	-	-	-	-	-	-
10^{-7}	-	1	1	1	1	1	1	1	1
10^{-8}	1	1	1	1	1	1	1	1	1
10^{-9}	1	1	1	1	1	1	1	1	1
10^{-10}	1	1	1	1	1	1	1	1	1

5.5 Cahn-Hilliard Equation

The fifth problem [33] is the Cahn-Hilliard equation (2.6); we considered it in Section 2.6:

$$u_t = -(0.001u_{xx} + u - u^3)_{xx}, \qquad -1 < x < 1, \qquad t > 0, \tag{5.10}$$

with the initial condition

$$u(x,0) = 0.1\cos(2\pi x) + 0.02\cos(10\pi x), \qquad -1 < x < 1,$$

and the boundary conditions

$$u_x(-1,t) = u_{xxx}(-1,t) = 0, \qquad t > 0,$$

$$u_x(1,t) = u_{xxx}(1,t) = 0, \qquad t > 0.$$

The converted system is:

$$\begin{cases} (u_1)_t = -10^{-3}(u_2)_{xx} - (u_1)_{xx} + 6(u_1)_x^2 u_1 + 3(u_1)^2(u_1)_{xx}, & -1 < x < 1, \quad t > 0, \\ 0 = (u_1)_{xx} - u_2, & -1 < x < 1, \qquad t > 0, \\ (u_1)_x(-1,t) = (u_1)_x(1,t) = 0, & t > 0, \\ (u_2)_x(-1,t) = (u_2)_x(1,t) = 0, & t > 0, \\ u_1(x,0) = 0.1\cos(2\pi x) + 0.02\cos(10\pi x), & -1 < x < 1, \\ u_2(x,0) = -0.4\pi^2\cos(2\pi x) - 2\pi^2\cos(10\pi x), & -1 < x < 1. \end{cases} \tag{5.11}$$

The approximate solution ($0 \leq t \leq 2.5$ and $-1 \leq x \leq 1$) is plotted in Figure 5.5 when $KCOL = 2$ and $atol = rtol = 10^{-6}$.

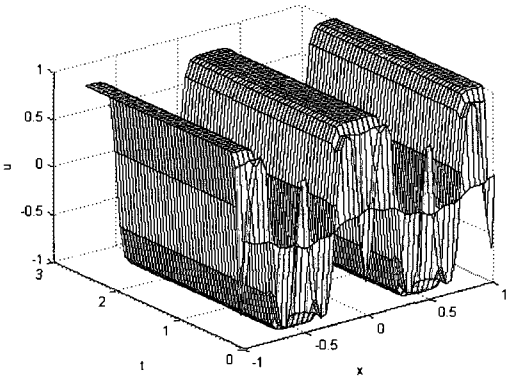

Figure 5.5: Approximate Solution of Example Five from BACOL42

In Figure 5.6, we can see the approximate solutions at $t = 0.05, 0.1, 0.3, 0.5$.

Table 5.13 displays how BACOL42 works with the different values of $KCOL$ (from 2 to 10) and tolerance (from 10^{-5} to 10^{-10}) for Example Five.

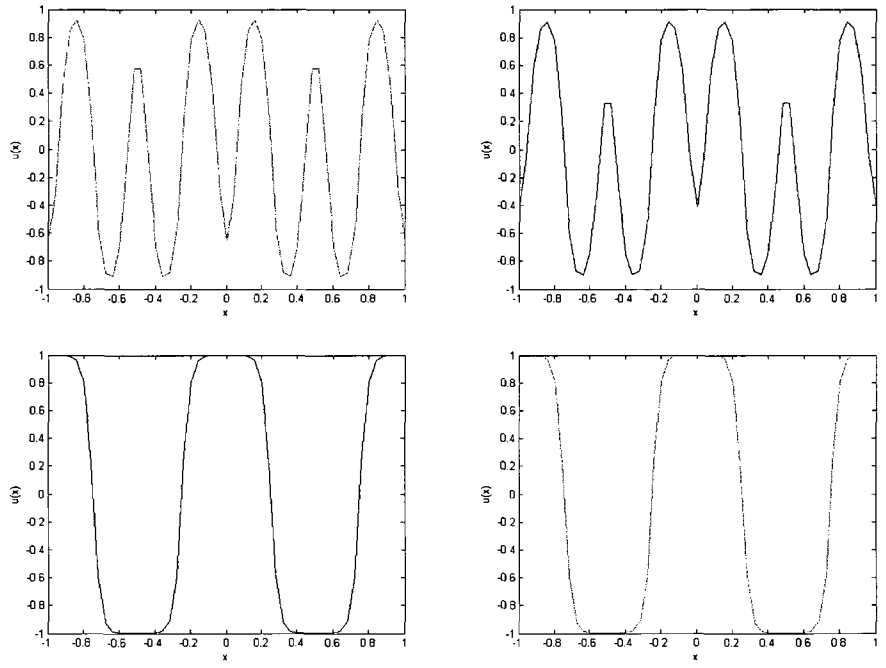

Figure 5.6: Approximate Solutions at $t = 0.05$(top left), 0.1(top right), 0.3(bottom left), 0.5(bottom right) from BACOL42 for Example Five

Table 5.13: Performance of BACOL42 with different $KCOL$ and tolerance values for Example Five

TOL	KCOL								
	2	3	4	5	6	7	8	9	10
10^{-5}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-6}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-7}	2	✓	✓	✓	✓	✓	✓	✓	✓
10^{-8}	1	1	✓	✓	✓	✓	✓	✓	✓
10^{-9}	1	1	1	1	1	1	1	1	1
10^{-10}	1	1	1	1	1	1	1	1	1

The ϵ version of (5.11) is

$$\begin{cases} (u_1)_t = -10^{-3}(u_2)_{xx} - (u_1)_{xx} + 6(u_1)_x^2 u_1 + 3(u_1)^2(u_1)_{xx}, & -1 < x < 1, \quad t > 0, \\ \epsilon(u_2)_t = (u_1)_{xx} - u_2, & -1 < x < 1, \qquad t > 0, \\ (u_1)_x(-1,t) = (u_1)_x(1,t) = 0, & t > 0, \\ (u_2)_x(-1,t) = (u_2)_x(1,t) = 0, & t > 0, \\ u_1(x,0) = 0.1\cos(2\pi x) + 0.02\cos(10\pi x), & -1 < x < 1, \\ u_2(x,0) = -0.4\pi^2\cos(2\pi x) - 2\pi^2\cos(10\pi x), & -1 < x < 1. \end{cases} \tag{5.12}$$

Table 5.14 displays how BACOL works with the different values of $KCOL$ (from 2 to 10) and tolerance (from 10^{-5} to 10^{-10}) for Example Five.

Table 5.14: Performance of BACOL with different $KCOL$ and tolerance values for Example Five

TOL	KCOL								
	2	3	4	5	6	7	8	9	10
10^{-5}	✓	✓	✓	✓	✓	✓	✓	✓	✓
10^{-6}	✓	✓	✓	✓	1	1	1	1	1
10^{-7}	1	✓	✓	1	1	1	✓	✓	✓
10^{-8}	3	-	✓	-	-	✓	✓	✓	✓
10^{-9}	-	-	-	-	-	-	-	-	-
10^{-10}	-	-	-	-	-	-	-	-	-

5.6 Conclusion

From Table 5.1, Table 5.5, Table 5.9, Table 5.11, and Table 5.13, we can see that BACOL42 can solve the problems with tolerances greater than or equal to 10^{-7}, and for some problems, the tolerance can be smaller than 10^{-7}. The range of the value of KCOL is from 2 to 10. From Table 5.3, 5.7, 5.10, 5.12, 5.14, we can conclude that BACOL can solve these five examples with tolerances greater than or equal to 10^{-5} with the range of KCOL from 2 to 10.

In Table 5.2 and Table 5.4, almost all the values of the global errors are a little greater than tolerance, but they are the same order of magnitude. The errors in the first two rows of Table 5.6 corresponding to $TOL = 10^{-5}$ and $TOL = 10^{-6}$ are less than the tolerance. When the tolerance is greater than $TOL = 10^{-6}$, the global errors become a little greater than the tolerances but is the same order of magnitude. All the error values in Table 5.8 are greater than the tolerances. We can conclude that BACOL42 is comparatively better than BACOL applied to the ϵ version of the problem. (We acknowledge that the appropriate choice of ϵ requires further investigation.)

Comparing Table 5.1, 5.5, 5.9, 5.11, 5.13 with Table 5.3, 5.7, 5.10, 5.12, 5.14, generally we can see that BACOL42 can solve problems to higher accuracy than BACOL can. For some cases, BACOL can solve the problem for a greater range of $KCOL$ values. These conclusions depend on how the value of ϵ affects the solution of the ϵ form of the converted system. We acknowledge that further analysis of this question is required before we can make more concrete comparisons between these

two approaches.

For each test problem, we provided a uniform initial mesh with $NINT = 10$. BACOL42 and BACOL then attempt to adapt the number of subintervals to meet the requested tolerances using as few subintervals as possible. The values of $NINT$ vary for different values of $KCOL$ and tolerance. The range of $NINT$ values we have seen varies from less than 10 to several hundred. Usually when the requested tolerance is smaller, $NINT$ will be relatively large. And $NINT$ values for larger $KCOL$ values (greater than 5) are usually smaller than for smaller $KCOL$ values. Also the difficulty of the problem can lead to the larger $NINT$ values.

For example, the following $NINT$ values were used for the final meshes ($atol = rtol = 10^{-6}$, $KCOL = 6$).

Problem Number	1	2	3	4	5
Final $NINT$	10	10	12	16	84

Chapter 6

Numerical Solution of Fourth Order PDEs with *pdepe* and MOVCOL4

The codes *pdepe* and MOVCOL4 do not have spatial error control and thus cannot be compared directly with codes that do provide spatial error control, such as BACOL42. A fundamental issue with *pdepe* and MOVCOL4 is that it is essentially impossible to know how many mesh points should be provided in order to obtain a desired accuracy. In the case of *pdepe* it is also impossible to know where the mesh points should be located; in fact it is likely that the locations of the mesh points would need to change with time, and *pdepe* is incapable of doing this, so in general one would need to provide an initial mesh that is as fine as it will need to be for the entire time interval. A detailed comparison of BACOL42 with *pdepe* or MOVCOL4 is therefore beyond

the scope of this thesis.

In this chapter, we provide numerical results for *pdepe* and MOVCOL4 to explore their use in the numerical solution of some fourth order PDEs. For *pdepe* and MOVCOL4, we choose the initial uniform mesh to have at least as many points as BACOL42 required to solve the same problem. After meeting this constraint, the choice of $NINT$ for *pdepe* and MOVCOL4 was made some what arbitrarily. (Since neither code performs spatial error control, the determinination of an optimal value of mesh points is actually somewhat difficult and such an investigation is beyond the scope of the thesis.) We will expect to see only approximate agreement with the error controlled results from BACOL42. The primary point of this chapter is to demonstrate the use of these other packages on some of the standard test problems.

We recall that MOVCOL4 can be applied directly to fourth order PDEs but that *pdepe* is designed to handle coupled systems of second order time-dependent PDEs and elliptic problems in 1-dimension. The converted system form (1.2) that we have employed for BACOL42 can also be described in this way. That is the equation $(u_1)_{xx} - u_2 = 0$ that we have called an ODE in space can also be called a 1D elliptic problem.

6.1 Example One

In this section, we apply *pdepe* and MOVCOL4 to Equation (2.12) discussed in Section 2.7.1. Recall that the PDE has the form

$$u_t = -u_{xxxx}.$$

- *pdepe*

The converted system form (5.1) we used for BACOL42 is the form that must be provided to *pdepe*. We chose the number of initial mesh points to be 21; when $atol = rtol = 10^{-6}$, $0 \leq x \leq \pi$ and $0 \leq t \leq 1$, the approximate solution is plotted in Figure 6.1.

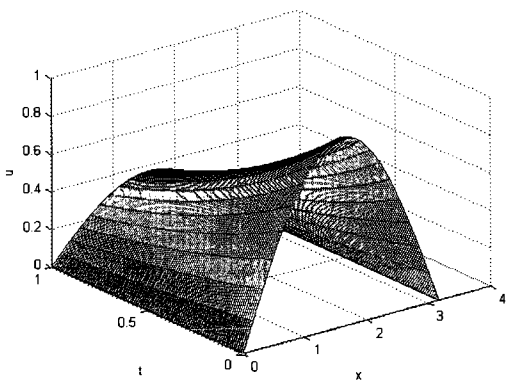

Figure 6.1: Approximate Solution of Example One from *pdepe*

- *MOVCOL4*

MOVCOL4 can solve equation (2.12) directly with 21 mesh points and the approximate solution is plotted in Figure 6.2 when $0 \leq x \leq \pi$, $0 \leq t \leq 1$, and

$atol = rtol = 10^{-6}$.

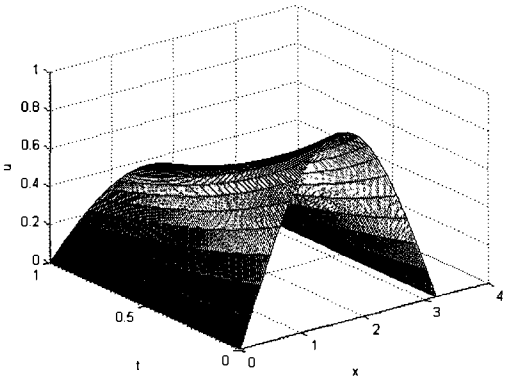

Figure 6.2: Approximate Solution of Example One from MOVCOL4

6.2 Example Two

This example (see Section 2.7.2) has the form

$$u_t = -\left(\frac{\sin t}{3 + \cos t}\right) u_{xxxx}.$$

- *pdepe*

We must use the converted system (5.3) in order to use the solver *pdepe*. When $atol = rtol = 10^{-6}$, $0 \leq x \leq \pi$ and $0 \leq t \leq 0.6$, 51 mesh points are employed to solve the problem. The approximate solution is plotted as Figure 6.3.

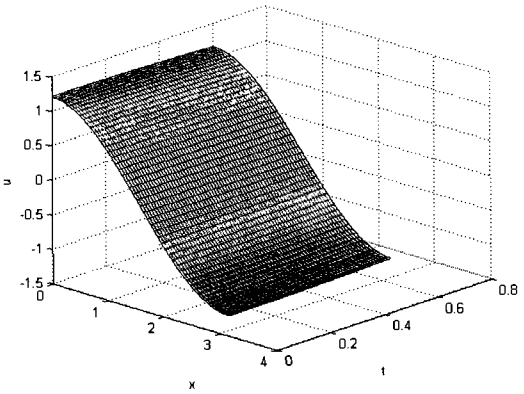

Figure 6.3: Approximate Solution of Example Two from *pdepe*

- *MOVCOL4*

With $atol = rtol = 10^{-6}$, we applied MOVCOL4 using 10 mesh points and the approximate solution ($0 \leq x \leq \pi$ and $0 \leq t \leq 0.6$) obtained is plotted in Figure 6.4.

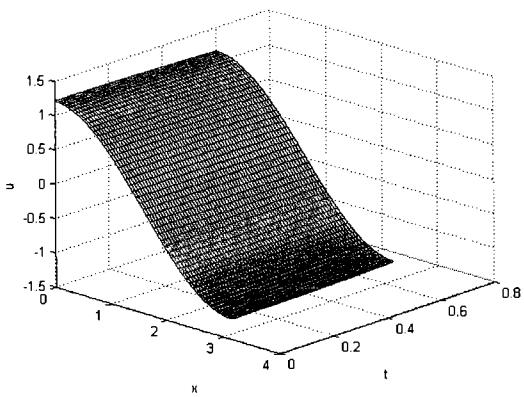

Figure 6.4: Approximate Solution of Example Two from MOVCOL4

6.3 Example Three

The third problem was introduced in Section 5.3. The equation has the following form,

$$u_t = -(u^{\frac{1}{2}} u_{xxx})_x.$$

- *pdepe*

The converted system of (5.5) is required in order to use the solver *pdepe*. When $atol = rtol = 10^{-6}$, $-1 \leq x \leq 1$, and $0 \leq t \leq 0.0007$, the approximate solution obtained is plotted in Figure 6.5. 101 mesh points were used.

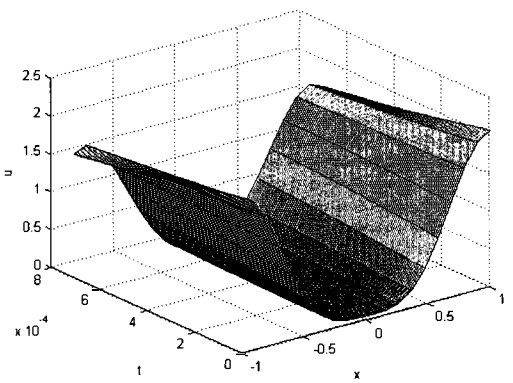

Figure 6.5: Approximate Solution of Example Three from *pdepe*

- *MOVCOL4*

We have *atol* and *rtol* both equal to 10^{-6}. We use the sample driver program provided with MOVCOL4 to solve the PDE starting with 257 uniformly spaced mesh points (This number of the mesh points is set in the driver program provided by the authors). The approximate solution we obtain is plotted in Figure 6.6 when $-1 \leq x \leq 1$ and $0 \leq t \leq 0.0007$.

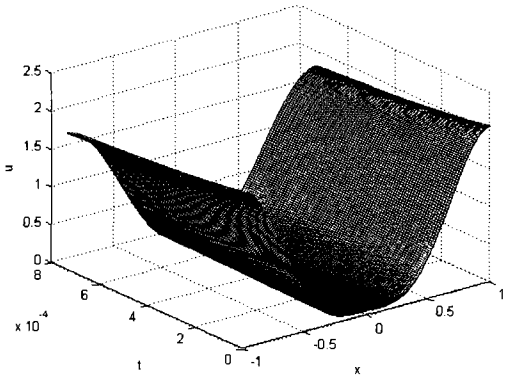

Figure 6.6: Approximate Solution of Example Three from MOVCOL4

6.4 Example Four

This example (see Section 5.4) has the form

$$u_t = -uu_x - u_{xx} - u_{xxxx}.$$

- *pdepe*

We rewrite the equation in the converted system form (5.8) in order to use the solver *pdepe*. When $atol = rtol = 10^{-6}$, $0 \leq x \leq 32\pi$ and $0 \leq t \leq 1$, 51 mesh points are employed to solve the problem. The approximate solution is plotted in Figure 6.7.

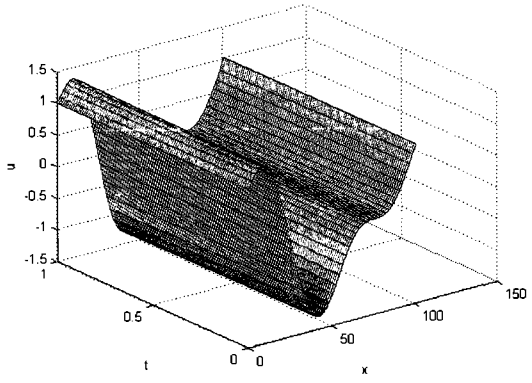

Figure 6.7: Approximate Solution of Example Four from *pdepe*

- *MOVCOL4*

With $atol = rtol = 10^{-6}$, we applied 51 mesh points to MOVCOL4, the approximate solution ($0 \leq x \leq 32\pi$ and $0 \leq t \leq 1$) obtained is plotted in Figure 6.8.

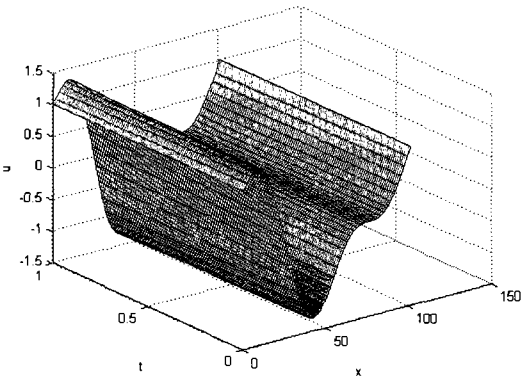

Figure 6.8: Approximate Solution of Example Four from MOVCOL4

6.5 Example Five

The fourth example which was described in Section 5.5 has the form

$$u_t = -(0.001u_{xx} + u - u^3)_{xx}.$$

When we tried to apply *pdepe* to solve this problem with several different choices for the initial number of (uniformly spaced) mesh points, such as 21, 51, 101 and 501, an error message was returned: "Warning: Failure at t=8.324973e-003. Unable to meet integration tolerances without reducing the step size below the smallest value allowed ".

- *MOVCOL4*

For this problem, a uniform initial mesh of 50 subintervals was employed; the numerical solution remains almost the same when the number of the mesh points is

increased from 21 to 101 [33]; the absolute tolerance and the relative tolerance both are 10^{-6}; the approximate solution is plotted in Figure 6.9 for $t = 1$.

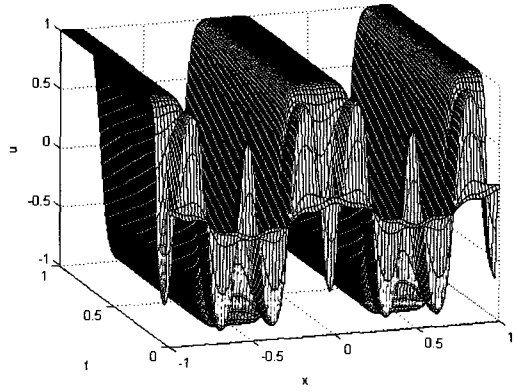

Figure 6.9: Approximate Solution of Example Five from MOVCOL4

In Figure 6.10, we can see the approximate solutions at $t = 0.05, 0.1, 0.3, 0.5$ more clearly.

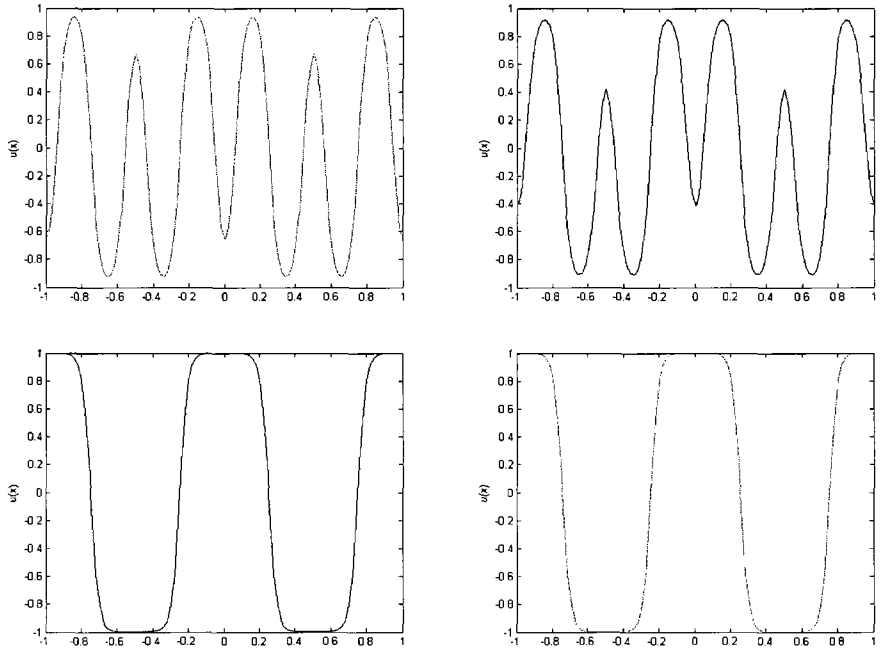

Figure 6.10: Approximate Solutions at $t = 0.05$ (top left), 0.1 (top right), 0.3 (bottom left), 0.5 (bottom right) from MOVCOL4 of Example Five

6.6 Summary

In this chapter, we have applied both *pdepe* and MOVCOL4 to solve five problems which were already solved by BACOL42 in Chapter 5. From Figure 6.1, Figure 6.2, and Figure 5.1, we can see that all three solutions are in close agreement. Similar comments hold for the second example as seen in Figure 6.3, Figure 6.4, and Figure 5.2, for the third example as seen in Figure 6.5, Figure 6.6, and Figure 5.3, and for the fourth example as seen in Figure 6.7, Figure 6.8 and Figure 5.4. Although *pdepe* cannot solve (5.10), Figure 6.9, 6.10, and Figure 5.5, Figure 5.6 of the fifth example show that MOVCOL4 and BACOL42 are in close agreement.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The thesis makes a number of contributions:

- We have provided a review of applications in which fourth order PDEs arise.
- We have reviewed standard algorithms for solving PDEs and surveyed a number of current popular PDE solvers.
- We have explored an approach that involves converting the fourth order PDE to a coupled system which contains one second order PDE and one second order ODE (in space). Two different treatments are discussed in the thesis: One approach involves the solution, using standard software (BACOL), of an approximate form of the converted system (the ϵ-version of the problem). The second approach treats the converted system directly and involved the development of BACOL42, an extension of BACOL that can solve fourth order parabolic PDEs

with adaptive error control. This approach also represents a first attempt at the development of a version of BACOL that can handle coupled PDE/ODE systems. We have considered five different test problems. The numerical results illustrate that BACOL42 can solve all test problems to tolerances greater than or equal to 10^{-7}, and the global error results show that BACOL42 does control the spatial error well.

- This thesis also provides a brief investigation of the application of four software packages (using three different approaches: the direct treatment of the fourth order problem, the direct treatment of the converted system, and the treatment of the approximate form of the converted system - the ϵ version of the converted system) for the numerical solution of several fourth order PDE test problems.

7.2 Future work

Three possible projects for future study are as follows:

- In some of the numerical results, we observed that BACOL42 can only solve problems to within a tolerance of 10^{-7}. We have not been able to get BACOL42 to work with very sharp tolerances. Despite the scaling we have introduced, it appears that there is still a conditioning issue. Future work involves further investigation of this issue.

- The current version of BACOL42 can only handle one fourth order parabolic PDE. However, some PDE models involve systems of fourth order PDEs or

fourth order PDEs coupled with second order PDEs. In addition, we also assume that there are two left boundary conditions and two right boundary conditions. When the number of boundary conditions is not the same at each end, BACOL42 cannot handle the problem. Therefore another example of future work is to extend the software to handle systems of fourth order PDEs without this restriction on the boundary conditions.

- Instead of solving the converted system, it would be interesting to see if one could develop a version of BACOL that can handle fourth order PDEs directly. The difficulty will be how to handle u_{xxx} and u_{xxxx}. The continuity of the B-spline representation of the approximate solution will have to increase from C^2 to C^4. This will change how the B-spline basis represents the approximate solution and it may impact on the ABD structure of the Newton matrices that arise.

- We use the approximate solution of the ϵ version of the converted system for comparison purposes, but we do not have a result relating the solution of the ϵ version of the converted system and to that of the original converted system. Thus another area for further investigation might be conduct an analysis to establish the relationship, as a function of ϵ, between

$$\begin{cases} (u_1)_t = f(x, t, u_1, (u_1)_x, (u_1)_{xx}, (u_2)_x, (u_2)_{xx}), \\ 0 = (u_1)_{xx} - u_2, \end{cases}$$

and

$$\begin{cases} (u_1)_t = f(t, x, u_1, (u_1)_x, (u_1)_{xx}, (u_2)_x, (u_2)_{xx}), \\ \epsilon(u_2)_t = (u_1)_{xx} - u_2, \end{cases}$$

including any possible boundary layers.

Appendix

In the appendix, we will give the detailed information about the modifications in the three subroutines: *CALJAC*, *CALRES*, and *DDASLV*.

1. *CALJAC*

The subroutine *CALJAC* computes the Jacobian matrix for the Newton iteration at the current time; that is, it computes PD= $\left(c_j A - \frac{\partial \tilde{F}}{\partial \underline{y}}\right)$, the matrix appearing in the left hand side of (4.3).

The first modification in *CALJAC* involves scaling the algebraic equations arising from collocating the ODE part of the converted system (1.2). The following code segment computes one *npde* $\times$ *npde* block of PD, corresponding to the collocation equations in one subinterval.

The original code which computes the values of $-\frac{\partial \tilde{F}}{\partial \underline{y}}$ for one subinterval is

```
                do 40 m = 1, npde
                   do 30 n = 1, npde

c                     nn is the pointer to the (n, m) element of the
c                     npde by npde submatrix.
                      nn = kk + (m-1)*npde*kcol + n

c                     mn is the pointer to the (n, m) element of dfdu.
                      mn = idfdu - 1 + (m - 1) * npde + n

c                     mn2 is the pointer to the (n, m) element of dfdux.
                      mn2 = mn + npde * npde

c                     mn3 is the pointer to the (n, m) element of dfduxx.
                      mn3 = mn2 + npde * npde
```

```
c                    now set up the derivative of F with respect to
c                    y at nn.
                     pd(nn) = - work(mn) * fbasis(jk)
     &                        - work(mn2) * fbasis(jk2)
     &                        - work(mn3) * fbasis(jk3)

   30             continue
   40          continue
```

$kk + 1$ is the pointer to the first element of one of the *npde* by *npde* submatrixes of PD. *idfdu* is a pointer to the work array where the $\frac{\partial f}{\partial u}$, $\frac{\partial f}{\partial u_x}$, and $\frac{\partial f}{\partial u_{xx}}$ values are stored. *fbasis* is an array where values of the B-spline functions (stored beginning at $fbasis(jk)$) and their first derivatives (stored beginning at $fbasis(jk2)$) and second derivatives (stored beginning at $fbasis(jk3)$) are stored.

In the new code, we need to perform the scaling indicated on the left hand side of (4.4). We are assuming $npde = 2$ and will treat the two inner loop iterates separately.

```
                do 40 m = 1, npde
c                   no need to use the inner loop, treat n=1 and n=2
c                   separately

c                     update of PD stays the same for
                      n=1
c                     nn is the pointer to the (n, m) element of the
c                     npde by npde submatrix.
                      nn = kk + (m-1)*npde*kcol + n

c                     mn is the pointer to the (n, m) element of dfdu.
                      mn = idfdu - 1 + (m - 1) * npde + n
```

```
c                    mn2 is the pointer to the (n, m) element of dfdux.
                     mn2 = mn + npde * npde

c                    mn3 is the pointer to the (n, m) element of dfduxx.
                     mn3 = mn2 + npde * npde

c                    now set up the derivative of F with respect to
c                    y at nn.
                     pd(nn) = - work(mn) * fbasis(jk)
     &                        - work(mn2) * fbasis(jk2)
     &                        - work(mn3) * fbasis(jk3)

c                    scale the left hand side by cj for
                     n=2
c                    nn is the pointer to the (n, m) element of the
c                    npde by npde submatrix.
                     nn = kk + (m-1)*npde*kcol + n

c                    mn is the pointer to the (n, m) element of dfdu.
                     mn = idfdu - 1 + (m - 1) * npde + n

c                    mn2 is the pointer to the (n, m) element of dfdux.
                     mn2 = mn + npde * npde

c                    mn3 is the pointer to the (n, m) element of dfduxx.
                     mn3 = mn2 + npde * npde

c                    now scale the derivative of  F with respect to
c                    y by cj at nn.
                     pd(nn) = - cj*(work(mn) * fbasis(jk)
     &                        + work(mn2) * fbasis(jk2)
     &                        + work(mn3) * fbasis(jk3))
  40           continue
```

The second modification of *CALJAC* is associated with the presence of zeros in the subblocks of the A matrix, corresponding to the ODE part of (1.2). We skip even rows in the update of PD.

The original code which adds $c_j A$ to $-\frac{\partial \bar{F}}{\partial \underline{y}}$ is

```
      call daxpy(nint*nsizbk, cj, abdblk, 1, pd(ipdblk), 1)
```

In the new *CALJAC* we must not add $c_j A$ to $-\frac{\partial \tilde{F}}{\partial \underline{y}}$ for the even rows of A. The modification is

```
c     skip every second row, because we must not add on cj*A
      call daxpy(nint*nsizbk/2, cj, abdblk, 2, pd(ipdblk), 2)
```

Note here that 2 is the increment (the last argument to *DAXPY*), so every second row in the update of PD is skipped.

2. *CALRES*

The subroutine *CALRES* generates the residual

$$\underline{G}(t, \underline{y}(t), \underline{y}'(t)) = A\underline{y}'(t) - \underline{\tilde{F}}(t, \underline{y}(t)) = 0,$$

at the current time, t. As indicated earlier, the general form of the Newton system is:

$$\left(c_j A - \frac{\partial \underline{\tilde{F}}}{\partial \underline{y}}\right) \triangle \underline{y}_{n+1}^{(m)} = - \left[A\left(\frac{\alpha}{h_{n+1}} \underline{y}_{n+1}^{(m)} + \beta\right) - \underline{\tilde{F}}(t_{n+1}, \underline{y}_{n+1}^{(m)})\right].$$

In *CALRES* the residual is stored in the array *delta* and *CALRES* must compute the right hand side of the above equation. The original code to calculate *delta* is

```
      do 70 i = 1, nint
         do 60 j = 1, kcol + nconti
            do 50 k = 1, kcol
               kk = 1+(i-1)*npde*npde*kcol*(kcol+nconti)
     &              +(j-1)*npde*npde*kcol+(k-1)*npde
               do 40 m = 1, npde
                  ii = npde+(i-1)*npde*kcol+(k-1)*npde+m
                  mm = (i-1)*kcol*npde+(j-1)*npde+m
                  delta(ii) = delta(ii) + abdblk(kk) * yprime(mm)
   40          continue
   50       continue
   60    continue
   70 continue
```

Note that before the calculation, *delta* contains evaluations of the function f at the collocation points; after the calculation, *delta* contains the residual of the DAE system; i.e., the right hand side of the Newton system given above. The array *abdblk* stores the elements of the *nint* blocks in the middle of the ABD collocation matrix, *yprime* is the derivative of y with respect to time at the current time, y is the vector of B-spline coefficients at the current time, and $nconti = 2$ is the number of continuity conditions at each internal mesh point. Because every second row of the middle part of matrix A is zero, we need to skip those lines when updating the residual. In BACOL42, we only consider one single fourth order PDE (npde=2), so we change the loop

```
               do 40 m = 1, npde
```

to

```
               do 40 m = 1, 1
```

so that only the first element in each subvector of *delta* is updated.

3. *DDASLV*

The subroutine *DDASLV* is part of the DDASSL package; it handles the linear systems arising in the Newton iteration performed by DDASSL. We perform the scaling on the left hand size of Equation (4.4) in *CALJAC*. In *DDASLV* we need to scale the right hand side of equation (4.4) by c_j, i.e., scale the residual corresponding to the ODE in order to improve the conditioning. The components of the residual corresponding to the boundary conditions were already scaled in the original BACOL.

This is the original code

```
c     kcol (numerical solution)

c     scale the right hand side of the Newton system corresponding
c     to the left boundary conditions
      call dscal(npde, cj, delta, 1)

c     scale the right hand side of the Newton system corresponding
c     to the right boundary conditions
      call dscal(npde, cj, delta(neq1-npde+1), 1)

c     solving the linear system
      call crslve(wm(npd), npde, 2*npde, wm(npdbk1), kcol*npde,
     &            (kcol+nconti)*npde, nint, wm(npdbt1), npde,
     &            iwm(lipvt), delta, 0)
c-----------------------------------------------------------------
c     kcol+1 (compute the error estimate)

      call dscal(npde, cj, delta(neq1+1), 1)
      call dscal(npde, cj, delta(neq-npde+1), 1)

      call crslve(wm(npdtp2), npde, 2*npde, wm(npdbk2),
     &            (kcol+1)*npde, (kcol+1+nconti)*npde, nint,
     &            wm(npdbt2), npde, iwm(lipvt2), delta(neq1+1), 0)
```

and after introducing scaling of the ODE collocation equations, we get

```
c     kcol (numerical solution)

c     scale the right hand side of the Newton system corresponding
c     to the left boundary conditions
      call dscal(npde, cj, delta, 1)

c     scale the right hand side of the Newton system corresponding
c     to the second equation
      call dscal(npde*kcol*nint/2, cj, delta(npde+2), 2)

      call dscal(npde, cj, delta(neq1-npde+1), 1)

c     solving the linear system
      call crslve(wm(npd), npde, 2*npde, wm(npdbk1), kcol*npde,
     &            (kcol+nconti)*npde, nint, wm(npdbt1), npde,
     &            iwm(lipvt), delta, 0)
c-----------------------------------------------------------------------
c     kcol+1 (compute the error estimate)

      call dscal(npde, cj, delta(neq1+1), 1)

c     scale the right hand side of the Newton system corresponding
c     to the second equation
      call dscal(npde*(kcol+1)*nint/2, cj, delta(neq1+4), 2)

      call dscal(npde, cj, delta(neq-npde+1), 1)

      call crslve(wm(npdtp2), npde, 2*npde, wm(npdbk2),
     &            (kcol+1)*npde, (kcol+1+nconti)*npde, nint,
     &            wm(npdbt2), npde, iwm(lipvt2), delta(neq1+1), 0)
```

Note that subroutine *dscal* scales *delta* by a constant *cj*. The last argument is the increment; it is 2 because we apply the scaling only to the even components of *delta*. The first and the last calls to *dscal* in both *kcol* case and $kcol + 1$ case represent scaling of boundary conditions, and are unchanged.

Bibliography

[1] A. Kassam, L. N.Trefethen, *Fourth-order time-stepping for stiff PDEs*, SIAM J. Sci. Comput. 26 (2005), pp. 1214-1233

[2] E. Beretta, M. Bertsch, and R. Dal Passo, *Nonnegative solutions of a fourth-order nonlinear degenerate parabolic equation*, Arch. Rat. Mech. Anal. 129 (1995), 175-200

[3] A. L. Bertozzi, *The mathematics of moving contact lines in thin liquid films*, Notices Amer. Math. Soc. 45 (1998), 689-697

[4] A. L. Bertozzi and J. B. Greer, *Low-curvature image simplifiers: global regularity of smooth solutions and laplacian limiting schemes*, Comm. Pure Appl. Math. 57 (2004) 764-790

[5] C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, New York (1978)

[6] J. W. Cahn and J. E. Hilliard, *Free energy of a nonuniform system. I. interfacial free energy*, J. Chem. Phys. 28 (1958), 258-267

[7] G. Carrero, M. J. Hendzel and G. de Vries, *Modelling the compartmentalization of splicing factors*, J. Theor. Biol. 239 (2006), 298-312

[8] W. Cheney and D. Kincaid, *Numerical Mathematics and Computing (Fifth Edition)*, Brooks/Cole, Pacific Grove, CA, (2004)

[9] P. Constantin, T. F. Dupont, R. E. Goldstein, L. P. Kadanoff, M. J. Shelley, and S. Zhou, *Droplet breakup in a model of the Hele-Shaw cell*, Phys. Rev. E 47 (1993), 4169-4181

[10] P. Coullet, C. Elphick, and D. Repaux, *Nature of spatial chaos*, Phys. Rev. Lett. 58 (1987), 431-434

[11] G. T. Dee and W. van Saarloos, *Bistable systems with propagating fronts leading to pattern formation*, Phys. Rev. Lett. 60 (1988), 2641-2644

[12] P. Degond, F. Mehats, and C. Ringhofer, *Quantum energy-transport and drift-diffusion models*, J. Stat. Phys. 118 (2005), 625-665

[13] J.C. Diaz, G. Fairweather and P. Keast, *FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination*, ACM Trans. Math. Softw. 9 (1983), 358-375

[14] A. Edena and V. K. Kalantarov, *The convective Cahn-Hilliard equation*, Appl. Math. Lett. 20 (2007), 455-461

[15] M. L. Frankel, *On the equation of a curved flame frone*, Physica D 30 (1988), 28-42

[16] M. L. Frankel, *On the nonlinear evolution of a solid-liquid interface*, Phys. Lett. A 128 (1988), 57-60

[17] C. F. Gerald and P. O. Wheatley, *Applied Numerical Analysis (Sixth Edition)*, Addision-Wesley, New York (1999)

[18] M. P. Gualdani, *Quantum models for semiconductors and nonlinear diffusion equations of fourth order*, Proceedings of the MSRJ Workshop, Women in Mathematics (2006), 143-148

[19] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*, Berlin, Heidelberg, New York, Tokyo, Springer-Verlag (1991), 550-555, *http : //www.unige.ch/ hairer/prog/stiff/radau5.f*

[20] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*, Springer-Verlag, New York (1991)

[21] W. Huang and R. D. Russell, *A moving collocation method for solving time dependent partial differential equations*, Appl. Num. Math. 20 (1996), 101-116

[22] A. Jüngel and S. Tang, *Global nonnegative solutions of a nonlinear fourth-order parabolic equation for quantum systems*, SIAM J. Math. Anal. 32 (2000), 760-777

[23] P. Keast and P. H. Muir, *Algorithm 688 EPDCOL: A more efficient PDECOL code*, ACM Trans. Math. Softw. 17 (1991), 153-166

[24] Y. Kuramoto and T. Tsuzuki, *Persistent propagation of concentration waves in dissipative media far from equilibrium*, Progr. Theoret. Phys. 55 (1976), 356-369

[25] J. D. Lambert, *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*, John Wiley & Sons, New York (1991)

[26] Q. Liu, Z. Yao and Y. Ke, *Solutions of fourth-order partial differential equations in a noise removal model*, J. Differential Equations (2007), 1-11

[27] M. Lysaker, A. Lundervold and X. C. Tai, *Noise removal using fourth-order partial differential equation with applications to medical magnetic resonance images in space and time*, IEEE Trans. Image Process. 12 (2003), 1579-1590

[28] N. K. Madsen and R. F. Sincovec, *Algorithm 540 PDECOL: General collocation software for partial differential equations*, ACM Trans. Math. Softw. 5 (1979), 326-351

[29] C. B. Moler, *Numerical Computing with MATLAB*, SIAM, Philadelphia, (2004)

[30] P. K. Moore, *Comparison of adaptive methods for one dimensional parabolic systems*, Appl. Num. Math. 16 (1995), 471-488

[31] P. K. Moore, *Interpolation error-based a posteriori error estimation for two-point boundary value problems and parabolic equations in one space dimension*, Numer. Math. 90 (2001), 149-177

[32] L. R. Petzold, *A Description of DASSL: A Differential/Algebraic System Solver*, Tech. Rep., Sandia Labs, Livermore, CA (1986)

[33] R. D. Russell, J. F. Williams and X. Xu, *MOVCOL4: A moving mesh code for fourth-order time-dependent partial differential equations*, SIAM J. Sci. Comput. 29 (2007), 197-220

[34] D. C. Sarocka and A. J. Bernoff, *An intrinsic equation of interfacial motion for the solidification of a pure hypercooled melt*, Physica D 85 (1995), 348-374

[35] R. Wang, P. Keast and P. Muir, *A comparison of adaptive software for 1D parabolic PDEs*, J. Comp. Appl. Math. 169 (2004), 127-150

[36] R. Wang, P. Keast and P. Muir, *BACOL: B-Spline Adaptive COLlocation Software for 1-D Parabolic PDEs*, ACM Trans. Math. Softw. 30 (2004), 454-470

[37] R. Wang, P. Keast and P. Muir, *Algorithm 874: BACOLR: Spatial and temporal error control software for PDEs based on high order adaptive collocation*, ACM Trans. Math. Softw. 34, (2008), 15:1-15:28

[38] W. W. Wang and J. R. Lister, *Similarity solutions for van der Waals rupture of a thin films on a solid substrate*, Phys. Fluids 11 (1999), 2454-2462

[39] T. P. Witelski, A. J. Bernoff and A. L.Bertozzi, *Blow-up and dissipation in a critical-case unstable thin film equation*, Euro. J. Appl. Math. 15 (2004), 223-256

[40] A. Vande Wouwer, P. Saucez and W. E. Schiesser, *Adaptive Method of Lines*, CRC Press, Boca Raton, (2001)

[41] M. Xu and S. L. Zhou, *Existence and uniqueness of weak solutions for a generalized thin film equation*, Nonlinear Anal. 60 (2005), 755-774

[42] M. Xu and S. L. Zhou, *Existence and uniqueness of weak solutions for a fourth-order nonlinear parabolic equation*, J. Math. Anal. Appl. 325 (2007), 636-654

[43] *http://en.wikipedia.org/wiki/Phosphorylation*

[44] *http://en.wikipedia.org/wiki/Dephosphorylation*

[45] *http://faculty.smu.edu/pmoore/hp4/hp4.html*