

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations, and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

**ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

**HIGH ORDER ADAPTIVE COLLOCATION SOFTWARE
FOR 1-D PARABOLIC PDES**

**By
Rong Wang**

**SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
DALHOUSIE UNIVERSITY
HALIFAX, NOVA SCOTIA
AUGUST, 2002**

© Copyright by Rong Wang, 2002



**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-75710-2

Canada

DALHOUSIE UNIVERSITY
FACULTY OF GRADUATE STUDIES

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled: "High Order Adaptive Collocation Software for 1-D Parabolic PDES" by Rong Wang, in partial fulfillment for the degree of Doctor of Philosophy.

Dated: August 28, 2002

External Examiner:


/ Dr. Joseph E. Flaherty /

Research Co-Supervisor:


Dr. Patrick Keast

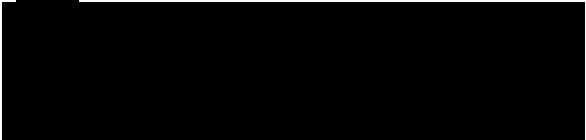
Research Co-Supervisor:


Dr. Paul Muir

Examining Committee:


Dr. Kay Spiteri


Dr. John Clements



DALHOUSIE UNIVERSITY

Date: August, 2002

Author: **Rong Wang**

Title: **HIGH ORDER ADAPTIVE COLLOCATION
SOFTWARE FOR 1-D PARABOLIC PDES**

Department: **Mathematics and Statistics**

Degree: **Ph.D.** Convocation: **October** Year: **2002**

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.


Signature of Author

THE AUTHOR RESERVES OTHER PUBLICATION RIGHTS, AND NEITHER THE THESIS NOR EXTENSIVE EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT THE AUTHOR'S WRITTEN PERMISSION.

THE AUTHOR ATTESTS THAT PERMISSION HAS BEEN OBTAINED FOR THE USE OF ANY COPYRIGHTED MATERIAL APPEARING IN THIS THESIS (OTHER THAN BRIEF EXCERPTS REQUIRING ONLY PROPER ACKNOWLEDGEMENT IN SCHOLARLY WRITING) AND THAT ALL SUCH USE IS CLEARLY ACKNOWLEDGED.

I dedicate this thesis
to
Fred and Miranda.

Contents

List of Tables	viii
List of Figures	ix
Acknowledgments	xi
Abstract	xii
1 Introduction	1
1.1 The Problem Class	1
1.2 Introduction to the Standard Method-of-Lines	3
1.3 Overview of the Thesis	7
2 Previous Work in Mesh Adaptation	8
2.1 The Equidistribution Principle	9
2.2 r-refinement	11
2.3 h- and hp-refinement	14
2.4 Balancing the Spatial and Time Errors	18
2.5 Other Adaptive Packages	20
3 Overview of the BACOL Algorithm	22
3.1 Spatial Discretization	23
3.1.1 B-spline Basis	23
3.1.2 Collocation at Gaussian Points	25

3.1.3	Treatment of the Boundary Conditions	28
3.2	Time Integration	29
3.2.1	Index of DAEs	30
3.2.2	A Short Description of BDF Methods for DAEs	31
3.2.3	Structure of the Iteration Matrix	33
3.2.4	Consistent Initial Conditions for the DAE System	38
3.2.5	Replacement of the Linear Algebra Solver	39
3.2.6	Restarting DASSL after a Remeshing	41
3.2.7	Treatment of Poorly Conditioned Jacobian matrices	43
3.2.8	Other Modifications to DASSL	46
3.3	Spatial Adaptivity	50
3.3.1	Spatial Error Estimation	50
3.3.2	The Remeshing Strategy	53
3.3.3	Cycle Avoidance	59
3.3.4	The Remeshing Algorithm	60
4	Description of the BACOL Software	62
4.1	A Description of all Subroutines	62
4.2	User Supplied Subroutines	66
4.3	Sample Program	67
4.4	Structure of BACOL	87
5	Numerical Experiments	95
5.1	Statement of Test Problems	96
5.1.1	Problem 1	96
5.1.2	Problem 2	98
5.1.3	Problem 3	99
5.1.4	Problem 4	101
5.1.5	Problem 5	102
5.1.6	Problem 6	104

5.1.7	Problem 7	108
5.2	Numerical Validation of BACOL	109
5.2.1	<i>TOL</i> and Global Error Proportionality	109
5.2.2	Relationship between <i>KCOL</i> and <i>CPU</i> time	110
5.2.3	Number of Steps VS. Number of Remeshings	112
5.3	Numerical Comparisons of MOL Packages	114
5.3.1	Problem 1	118
5.3.2	Problem 2	121
5.3.3	Problem 4	124
5.3.4	Problem 6	125
5.3.5	Problem 7	127
5.4	Inconsistent Initial and Boundary Conditions	128
5.5	BACOL for Problems with Blow-Up Solutions	131
5.6	Numerical Investigation of Convergence Rates	133
6	Conclusions and Future Work	137
	Bibliography	139

List of Tables

5.1	L^2 -norm error for Problem 1 with $\epsilon = 10^{-3}$.	110
5.2	L^2 -norm error for Problem 1 with $\epsilon = 10^{-4}$.	110
5.3	<i>CPU</i> time for Problem 1 with $\epsilon = 10^{-3}$.	111
5.4	<i>CPU</i> time for Problem 1 with $\epsilon = 10^{-4}$.	111
5.5	NS/NR for Problem 1 with $\epsilon = 10^{-3}$.	113
5.6	NS/NR for Problem 1 with $\epsilon = 10^{-4}$.	113
5.7	The maximum value of the solution as t approaches the blow-up time	133
5.8	Confirmation of superconvergence.	136

List of Figures

3.1	Structure of $\frac{\partial F}{\partial y'}$	35
3.2	Structure of $\frac{\partial F}{\partial y}$	36
3.3	Structure of the iteration matrix in BACOL	38
3.4	Structure of the iteration matrix for $p = 4$	40
3.5	Structure of the iteration matrix for $p = 4$	41
4.1	Structure of BACOL software	88
4.2	Percentage of the <i>CPU</i> time in main subroutines: single equation and before modifications to DDANRM.	90
4.3	Percentage of the <i>CPU</i> time in main subroutines: single equation and after modifications to DDANRM.	91
4.4	Percentage of the <i>CPU</i> time in main subroutines: 4 equations and before modifications to DDANRM.	93
4.5	Percentage of the <i>CPU</i> time in main subroutines: 4 equations and after modifications to DDANRM.	94
5.1	Problem 1 for $\epsilon = 10^{-4}$	98
5.2	Problem 2 for $\epsilon = 10^{-4}$	99
5.3	Problem 3 for $\epsilon = 10^{-6}$	100
5.4	Problem 4 for $\epsilon = 10^{-6}$	101
5.5	The step function.	102
5.6	Problem 5 for $u(x, t)$	103
5.7	Problem 5 for $v(x, t)$	104

5.8	Problem 6 for $u_1(x, t)$.	106
5.9	Problem 6 for $u_2(x, t)$.	106
5.10	Problem 6 for $u_3(x, t)$.	107
5.11	Problem 6 for $u_4(x, t)$.	107
5.12	Problem 7 for $u(x, t)$.	109
5.13	The <i>CPU</i> time and the error for Problem 1 with $\epsilon = 10^{-3}$.	119
5.14	The <i>CPU</i> time and the error for Problem 1 with $\epsilon = 10^{-4}$.	120
5.15	The <i>CPU</i> time and the error for Problem 2 with $\epsilon = 10^{-3}$.	122
5.16	The <i>CPU</i> time and the error for Problem 2 with $\epsilon = 10^{-4}$.	123
5.17	The <i>CPU</i> time and the error for Problem 4.	124
5.18	The <i>CPU</i> time and the error for Problem 6.	126
5.19	The <i>CPU</i> time and the error for Problem 7.	127
5.20	The approximate solutions using BACOL and EPDCOL.	129
5.21	The <i>CPU</i> time and the error for Problem 5.	131
5.22	$U(x, t)/U(0.5, t)$ at $t = 0, 0.08, 0.0824, 0.082437$.	132

Acknowledgments

I would like to thank to my supervisors, Dr. Pat Keast and Dr. Paul Muir, for giving me a lot of valuable comments. I am also thankful to Dr. Raymond Spiteri, Dr. John Clements and Dr. Joe Flaherty for some helpful comments and suggestions.

Abstract

In this thesis a high order adaptive method-of-lines package, BACOL, is developed for solving one dimensional parabolic partial differential equations. Collocation with a B-spline basis is used for the spatial discretization. An approximate solution is calculated in a degree p piecewise polynomial subspace, and the spatial error estimate is obtained by using a second solution computed in a degree $p+1$ piecewise polynomial subspace.

BACOL controls both the spatial error and the temporal error. After each time step the spatial error is estimated, and if it is larger than the spatial error tolerance, an equidistribution principle is employed to redistribute the mesh. At the same time, the number of mesh points employed can be changed if necessary. The time integration is done by a differential-algebraic-equation (DAE) solver, DASSL, which uses backward differentiation formulas. Modifications to DASSL included replacing the original linear system solver by the almost block diagonal system solver, COLROW; scaling the Newton iteration matrix to avoid the large condition number generated by the index-1 DAEs; and changing the dimension of tolerance from the number of DAEs to the number of PDEs. Computational results indicate that BACOL is reliable and extremely efficient in dealing with problems having solutions with rapid variation.

Chapter 1

Introduction

Partial differential equations (PDEs) arise in a wide variety of physical problems. (See [43] for examples.) Roughly speaking, they can be separated into three categories: parabolic (e.g., the heat equation $u_t = u_{xx}$), hyperbolic (e.g., the wave equation $u_t = u_x$), elliptic (e.g., Laplace's equation $u_{xx} + u_{yy} = 0$). There are also many examples of PDEs with high order spatial or time derivatives (e.g., the Korteweg-de Vries equation $u_t + 6uu_x + u_{xxx} = 0$), and of equations which fall into none of the three general classifications (e.g. $u_{tt} = a(x, t)u_{xx}$, where $a(x, t)$ may be negative in some regions, and positive in others). Since the PDE models are usually nonlinear and typically do not have exact solutions, numerical algorithms are generally employed to obtain approximate solutions.

1.1 The Problem Class

In this thesis we will discuss systems of one dimensional (1-D) time-dependent parabolic PDEs of the form

$$u_t(x, t) = f(t, x, u(x, t), u_x(x, t), u_{xx}(x, t)), \quad x_a \leq x \leq x_b, \quad t \geq t_0, \quad (1.1)$$

where the initial condition is given by

$$u(x, t_0) = u_0(x), \quad x_a \leq x \leq x_b, \quad (1.2)$$

and the separated boundary conditions (BCs) are given by

$$b_L(t, u(x_a, t), u_x(x_a, t)) = 0, \quad t \geq t_0, \quad (1.3)$$

$$b_R(t, u(x_b, t), u_x(x_b, t)) = 0, \quad t \geq t_0, \quad (1.4)$$

where

$$\begin{aligned} u &= (u_1, u_2, \dots, u_{NPDE}), & u_t &= \left(\frac{\partial u_1}{\partial t}, \frac{\partial u_2}{\partial t}, \dots, \frac{\partial u_{NPDE}}{\partial t} \right), \\ u_x &= \left(\frac{\partial u_1}{\partial x}, \frac{\partial u_2}{\partial x}, \dots, \frac{\partial u_{NPDE}}{\partial x} \right), & u_{xx} &= \left(\frac{\partial^2 u_1}{\partial x^2}, \frac{\partial^2 u_2}{\partial x^2}, \dots, \frac{\partial^2 u_{NPDE}}{\partial x^2} \right), \\ f &= (f_1, f_2, \dots, f_{NPDE}), & u_0 &= (u_{0,1}, u_{0,2}, \dots, u_{0,NPDE}), \\ b_L &= (b_{L,1}, b_{L,2}, \dots, b_{L,NPDE}), & b_R &= (b_{R,1}, b_{R,2}, \dots, b_{R,NPDE}), \end{aligned}$$

where $NPDE$ is the number of PDEs. We require that the u_{xx} term does not vanish in the right-hand side of the PDEs.

Sometimes BCs arise in the nonseparated form

$$B(t, u(x_a, t), u_x(x_a, t), u(x_b, t), u_x(x_b, t)) = 0, \quad t \geq t_0, \quad (1.5)$$

where $B = (B_1, B_2, \dots, B_{2NPDE})$. We now briefly describe a general way to convert a PDE system with nonseparated BCs to one with separated BCs based on the discussion in [9]. Choose some $x_c \in (x_a, x_b)$ and let

$$\begin{aligned} w(s, t) &= w\left(\frac{x - x_a}{x_c - x_a}, t\right) = u(x, t), \quad x_a \leq x \leq x_c, \\ v(s, t) &= v\left(\frac{x_b - x}{x_b - x_c}, t\right) = u(x, t), \quad x_c \leq x \leq x_b. \end{aligned}$$

Then we obtain the following PDE system which is equivalent to (1.1), (1.2), (1.5),

$$\begin{aligned} w_t(s, t) &= f(t, s(x_c - x_a) + x_a, w(s, t), \frac{1}{x_c - x_a} w_s(s, t), \frac{1}{(x_c - x_a)^2} w_{ss}(s, t)), \\ v_t(s, t) &= f(t, s(x_c - x_b) + x_b, v(s, t), -\frac{1}{x_b - x_c} v_s(s, t), \frac{1}{(x_b - x_c)^2} v_{ss}(s, t)), \end{aligned}$$

for $0 \leq s \leq 1, t > t_0$. The initial conditions become

$$\begin{aligned} w(s, t_0) &= u_0(s(x_c - x_a) + x_a), & 0 \leq s \leq 1, \\ v(s, t_0) &= u_0(s(x_c - x_b) + x_b), & 0 \leq s \leq 1, \end{aligned}$$

and the boundary conditions, which are now *separated* become

$$B(t, w(0, t), \frac{1}{x_c - x_a} w_s(0, t), v(0, t), -\frac{1}{x_b - x_c} v_s(0, t)) = 0, \quad t \geq t_0,$$

$$w(1, t) = v(1, t), \quad \frac{1}{x_c - x_a} w_s(1, t) = -\frac{1}{x_b - x_c} v_s(1, t), \quad t \geq t_0.$$

We note that the size of the PDE system is doubled after the transformation. Thus the PDE system with a nonseparated boundaries needs more computational effort than the same PDE system with separated BCs. For further details of the technique we refer the reader to [9], which gives a discussion of this technique in the boundary value ODE case.

The spatial domain for our problem class is $[x_a, x_b]$. However, the linear transformation $T_1: p = \frac{x-x_a}{x_b-x_a}$ maps $x \in [x_a, x_b]$ to $p \in [0, 1]$. Similarly, the linear transformation $T_2: q = t - t_0$ maps $t \in [t_0, \infty]$ to $q \in [0, \infty]$. Therefore, without loss of generality, we will assume $x_a = 0$, $x_b = 1$, and $t_0 = 0$.

1.2 Introduction to the Standard Method-of-Lines

The numerical method of lines (MOL) is an approach for the treatment of time-dependent PDEs. The standard MOL involves two steps. The first step is the spatial discretization, in which the spatial derivatives are approximated by using, for example, finite difference methods [56], finite element methods [31], or collocation methods. We will consider a simple example shortly. This transforms the PDE system into a large set of initial value ordinary differential equations (ODEs), which are stiff in many cases. For a detailed discussion of stiffness, we refer the reader to [35]. The second step is the time integration, in which the system of ODEs is integrated in time by some standard ODE solver (e.g., GEARB [37]). Backward differentiation formula (BDF) methods and implicit Runge-Kutta methods are generally employed because of the stiffness of ODE systems.

In the early days of ODE software development, the solvers could only handle pure ODE systems. This represented a difficulty within the MOL context because the boundary conditions are usually algebraic equations rather than differential equations in time. There are two possibilities for dealing with the boundary conditions. The first approach is to differentiate the boundary conditions with respect to time and couple them with the large ODE system which arises from the discretization of the PDEs; then the coupled ODE system is treated in the usual way by an ODE solver. (See, e.g., PDECOL [53].) The second approach is to ask the user to incorporate the boundary conditions into the subroutine that defines the right-hand side of the PDEs (1.1) (See, e.g., DSS/2 [66]). About twenty years ago, there began to appear solvers which were able to handle ODEs coupled with algebraic constraints, i.e., differential-algebraic equations (DAEs). An example of such a solver is DASSL [61]. With the availability of such solvers, the boundary conditions can be treated directly and coupled with the ODEs.

The MOL is widely used in general-purpose PDE software largely because of the availability of high-quality ODE and DAE solvers. Most of the modern ODE solvers employ adaptive time integration, i.e., variable time step sizes and possibly variation of the order of integration formula. The fundamental assumption, which distinguishes the *standard* MOL approach from more sophisticated approaches developed later, is that the spatial domain is partitioned by a mesh which remains fixed throughout the entire integration. Adaptive error control is limited to the time dimension through the numerical solution of the ODE/DAE system. Therefore this is useful only when time integration error dominates spatial discretization error.

We will now briefly describe the spatial discretization process in the standard MOL. Let us consider the example PDE,

$$u_t = u_{xx}, \quad 0 \leq x \leq 1, \quad t \geq 0.$$

Assume that a uniform mesh $\{x_i\}_{i=0}^N$ partitions the x domain; i.e., $x_i = ih$, where

$h = 1/N$. Suppose that $U_i(t)$ approximates solution for $u(x_i, t)$ for $i = 0, \dots, N$. We can then employ a central finite difference formula for the spatial discretization, approximating $u_{xx}(x_i, t)$ by $\frac{U_{i+1}(t) - 2U_i(t) + U_{i-1}(t)}{h^2}$. Thus we have transformed the PDE into a system of ODEs of the form

$$\frac{d}{dt}U_i(t) = \frac{U_{i+1}(t) - 2U_i(t) + U_{i-1}(t)}{h^2},$$

with $i = 1, \dots, N - 1$. Assuming that the boundary conditions, which are usually algebraic constraints, are differentiated with respect to time to give ODEs which are then coupled with the ODE system arising from the discretization of PDEs, we can then apply a standard ODE solver for the time integration of this system. For a detailed introduction to the standard MOL, please refer to, e.g., [67].

In the 1970s, several general-purpose solvers, based on the MOL, for 1-D parabolic systems were developed.

- DSS/2 written by Schiesser [66] is a modification of an earlier code LEANS [65]. Centered and noncentered differences of even orders 2 through 10 are used for the spatial discretization. The time integration is done by GEARB ([37]).
- FORSIM was developed by Carver [29]. The spatial derivatives can be approximated by centered or noncentered difference schemes of various orders. The code allows for time integration of the ODE system using Runge-Kutta methods or BDF methods.
- MOL1D by Hyman [44] is available as PDELIB at ARNO (See GAMS [20], <http://gams.nist.gov>). The spatial discretization is accomplished by second-, fourth-, or sixth-order centered differences or unsymmetric second- or third-order differences. GEARB is employed for the time integration.
- PDEPACK was produced by Madsen and Sincovec [51]. Central finite differences are used for the spatial discretization. The time integrator is a modification of GEARB.

- PDECOL was also developed by Madsen and Sincovec [52, 53]. In contrast to the previous approach, a collocation method, using B-splines as the piecewise polynomial basis, is applied instead of finite difference methods. The degree of piecewise polynomial can be chosen to be from 3 to 19. The system of ODEs is now implicit, and GEARIB ([36]), a modification of GEARB, is employed to solve the resulting ODE system.

For more details about the early MOL approach, please refer to the survey paper by Machura and Sweet [50]. Further work involving the standard MOL approach has been conducted over the last twenty years. Some important work is

- Several software packages based on an early version of SPRINT (due to Berzins [12]) were developed for the NAG library. These are D03PCF, D03PDF, D03PEF, D03PHF, D03PJF, and D03PKF. For the spatial discretization D03PCF uses a central finite difference method, while D03PDF uses a Chebyshev collocation method, and the others apply the Keller box scheme, which is a second order finite difference method. The resulting ODE system is solved using a BDF method. D03PHF, D03PJF, and D03PKF can solve a parabolic system coupled with ODEs.
- EPDCOL by Keast and Muir [46] is a modification of PDECOL. It takes advantage of the structure of the matrix arising in the Newton iteration. The authors observe that the Newton iteration matrix has an almost block diagonal (ABD) form [30] and therefore replace the original banded linear system solver by COLROW, an ABD solver. This saves storage and improves the efficiency of PDECOL.
- PDECHEB [11] by Berzins and Dew implements a spatial discretization using a global element method, which is similar to a C^0 collocation method. Piecewise Chebyshev polynomials with C^0 continuity are employed as the spatial basis. PDECHEB can handle a wide range of elliptic-parabolic PDEs. The resulting

differential algebraic equation (DAE) system is solved by a modification of the DAE solver DASSL.

The standard MOL approach is efficient for problems with spatially smooth solutions. But since it fixes the mesh points for the entire computation, it becomes inefficient for problems with solutions of rapid spatial variation, such as narrow moving fronts. In such situations, we want to move the mesh points so that they will concentrate in and follow the regions where the solution changes rapidly. This is the so-called adaptive MOL approach, in the sense of both temporal (automatically adjusted time step sizes) and spatial (automatically adjusted spatial nodes) adaptivity.

1.3 Overview of the Thesis

In this thesis, a general-purpose adaptive MOL software, BACOL, for 1-D parabolic systems is developed. Collocation with a B-spline basis is used for the spatial discretization. An approximate solution is calculated in a degree p piecewise polynomial subspace, and the spatial error estimate is obtained by using a second solution computed in a degree $p + 1$ piecewise polynomial subspace. The remeshing strategy is developed based on the equidistribution principle. A modification of DASSL is used for the time integration. In chapter 2, we will review the previous work by other authors in the area of adaptive MOL. Chapter 3 describes all the algorithms employed in BACOL, including the spatial error estimate, the mesh redistribution algorithm, a technique to avoid large condition numbers associated with the DAEs, etc. In chapter 4 detailed descriptions of subroutines which make up BACOL are given. The descriptions of the necessary user-supplied subroutines are also given there. Chapter 5 gives the numerical experiments which demonstrate the capabilities of BACOL and comparisonis with some well-known codes. Chapter 6 gives our conclusions and suggestions for possible future projects.

Chapter 2

Previous Work in Mesh Adaptation

In the MOL approach one naturally obtains temporal error control through the adaptive stepsize selection algorithm included in the ODE/DAE software one applies to the ODE/DAE system after a spatial discretization of PDE system is performed. In the past twenty years, adaptive methods for the spatial discretization of PDE system have drawn considerable attention. The use of adaptive mesh techniques provides a far more efficient way to treat problems with rapidly varying solutions.

The three most common mesh adaptation strategies are:

- mesh motion (r-refinement),
- spatial mesh refinement or coarsening (h-refinement),
- order variation (p-refinement).

(Another adaptive approach involves employing different “upwinded” spatial discretization schemes for each mesh subinterval based on the estimate of the smoothness of the solution; those are “essentially non-oscillating” (ENO) and “weighted essentially non-oscillating” (WENO) schemes; See e.g., [68]. However, these are commonly used for resolving sharp fronts on a fixed mesh for hyperbolic PDE systems.)

Combining two or three different strategies, we can generate more sophisticated techniques, e.g., hp-refinement. In case of time-dependent PDEs (e.g., parabolic or hyperbolic PDEs), p-refinement, to our knowledge, is never applied alone. We will consider r-, h-, and hp-refinement.

Most methods adapt the nodes to equidistribute some error measure; this is called the equidistribution principle (EP). Therefore, we will first give a simple introduction to the EP in this chapter. We will then review past work in r-, h-, and hp-refinement. This will be followed by a discussion of the balancing of spatial and temporal errors. Finally we will give a survey of current existing adaptive software based on MOL for 1-D parabolic partial differential equations.

2.1 The Equidistribution Principle

The equidistribution principle was first introduced by White [72] for two-point boundary value ODEs. Here we will describe the basic idea of the EP using a sample problem in parabolic PDE form. For convenience, we will consider a parabolic system with Dirichlet boundary conditions,

$$u_t(x, t) = f(t, x, u(x, t), u_x(x, t), u_{xx}(x, t)), \quad 0 \leq x \leq 1, \quad t \geq 0, \quad (2.1)$$

$$u(x, 0) = u_0(x), \quad 0 \leq x \leq 1, \quad (2.2)$$

$$u(0, t) = 0, \quad u(1, t) = 0, \quad t \geq 0, \quad (2.3)$$

Assume that we have a uniform mesh $\{x_i\}_{i=0}^N$, with $x_i = ih$, where $h = 1/N$. Let $U_i(t)$ be the approximate solution for $u(x_i, t)$ for $i = 0, \dots, N$. If central finite difference formulas are employed for the spatial discretization, we have

$$u_x(x_i, t) \approx \frac{U_{i+1}(t) - U_{i-1}(t)}{2h}, \quad (2.4)$$

$$u_{xx}(x_i, t) \approx \frac{U_{i+1}(t) - 2U_i(t) + U_{i-1}(t)}{h^2}. \quad (2.5)$$

Substituting (2.4) and (2.5) into (2.1), we obtain

$$\frac{d}{dt}U_i(t) = f\left(t, x_i, U_i(t), \frac{U_{i+1}(t) - U_{i-1}(t)}{2h}, \frac{U_{i+1}(t) - 2U_i(t) + U_{i-1}(t)}{h^2}\right), \quad (2.6)$$

with $i = 1, \dots, N-1$. Applying the Dirichlet boundary conditions, $U_0(t) = U_N(t) = 0$, we can then solve the system of ODEs (2.6) for U_i , $i = 1, \dots, N-1$.

The equidistribution principle attempts to position the mesh points such that some measure of the spatial error is equally distributed over the subintervals. Roughly speaking, there are two strategies commonly used to apply the EP. The first one is to apply the EP in an integral form; this is employed in most h- or hp-refinement approaches. A popular choice is the arclength function for representing an idea of error distribution. In this case, the EP will require that the new mesh points, $\{x_i^*\}_{i=1}^K$, satisfy, for a fixed t ,

$$\int_{x_{i-1}^*}^{x_i^*} \sqrt{1 + (U_x(x, t))^2} dx = \text{constant},$$

where $i = 1, \dots, K$ and K does not necessarily equal N , i.e., the number of mesh points can be changed for the next mesh redistribution. Here we call $\sqrt{1 + (U_x(x, t))^2}$ the *monitor function*, and call the above mesh redistribution process a *remeshing*.

The second strategy is to apply the EP in a differential form; this is used in the r-refinement approaches. In r-refinement schemes, each mesh point is considered to be a function of time while the number of mesh points is fixed. Again assuming the arclength monitor function, let

$$E_i(t) = \int_{x_{i-1}(t)}^{x_i(t)} \sqrt{1 + (U_x(x, t))^2} dx,$$

where $i = 1, \dots, N$ and

$$\bar{E}(t) = \frac{1}{N} \int_0^1 \sqrt{1 + (U_x(x, t))^2} dx.$$

We see that $E_i(t)$ is used to give an idea of the error for the subinterval $[x_{i-1}(t), x_i(t)]$ and $\bar{E}(t)$ gives an idea of the average error. In order to move $x_{i-1}(t)$ and $x_i(t)$ closer

to each other when $E_i(t)$ is larger than $\bar{E}(t)$ and farther apart when $E_i(t)$ is smaller than $\bar{E}(t)$, we require

$$\frac{d}{dt}x_i(t) - \frac{d}{dt}x_{i-1}(t) = -\lambda(E_i(t) - \bar{E}(t)),$$

where $i = 1, \dots, N$, and λ is a proportionality factor. The term $\bar{E}(t)$ is eliminated by considering the corresponding equations for $[x_i(t), x_{i+1}(t)]$ and subtracting the first from the second. We then obtain

$$\frac{d}{dt}x_{i+1}(t) - 2\frac{d}{dt}x_i(t) + \frac{d}{dt}x_{i-1}(t) = -\lambda(E_{i+1}(t) - E_i(t)), \quad (2.7)$$

where $i = 1, \dots, N - 1$. Coupling the mesh equations (2.7) with the ODEs obtained from the spatial discretization of the original PDEs and the boundary conditions, we can then employ a standard ODE or DAE solver for the time integration. We will discuss the choice of λ in Section 2.2.

2.2 r-refinement

In the r-refinement approach, as mentioned in the previous subsection, the mesh points are considered to be time-dependent functions. After the spatial discretization step, the MOL gives a system of ODEs for the solution, which are coupled with a system of ODEs controlling the mesh selection. The node positions and the corresponding solutions are then calculated simultaneously. We note that the standard r-refinement approach employs a fixed number of mesh points and therefore spatial error control is not possible. Also, the problem becomes nonlinear even if original problem is linear.

One of the first papers on r-refinement was written by Miller and Miller [54, 55]. They employ finite elements on the mesh, $0 = x_0(t) < x_1(t) < \dots < x_{N-1}(t) < x_N(t) = 1$. In subsequent work, Adjerid and Flaherty [2, 3] obtained a “moving finite element” method by applying the EP to a spatial error estimate. An approximate solution

$U(x, t)$ is calculated in a piecewise linear polynomial subspace, i.e.

$$U(x, t) = \sum_{i=0}^N U_i(t) \Phi_i(x, t), \quad (2.8)$$

where

$$\Phi_i(x, t) = \begin{cases} \frac{x - x_{i-1}(t)}{x_i(t) - x_{i-1}(t)}, & x_{i-1}(t) \leq x \leq x_i(t), \\ \frac{x_{i+1}(t) - x}{x_{i+1}(t) - x_i(t)}, & x_i(t) \leq x \leq x_{i+1}(t), \\ 0, & \text{otherwise,} \end{cases} \quad (2.9)$$

and $U_i(t) \approx u(x_i(t), t)$. The spatial error is estimated in terms of a piecewise quadratic function $E(x, t)$ obtained as follows. A finite element method is applied to the PDEs on a piecewise linear subspace to obtain $U(x, t)$; then an improved solution $\hat{U}(x, t)$ is obtained by applying the finite element method on a piecewise quadratic subspace. We write $\hat{U}(x, t) = U(x, t) + E(x, t)$, and thus obtain $E(x, t)$ from $\hat{U}(x, t) - U(x, t)$. Let $\|\cdot\|_1$ denote the H^1 -norm in the interval $[a, b]$, i.e.,

$$\|f(x)\|_1 = \sqrt{\int_a^b (f^2(x) + f_x^2(x)) dx}.$$

The mesh points are determined from the ODE system

$$\dot{x}_i(t) - \dot{x}_{i-1}(t) = -\lambda(\|E_i(t)\|_1 - \|\bar{E}(t)\|_1), \quad i = 1, 2, \dots, N,$$

where λ is a proportionality factor, $\|E_i(t)\|_1$ is the H^1 -norm error for the i -th subinterval and $\|\bar{E}(t)\|_1 = \frac{1}{N}\|E(x, t)\|_1$ is the average error in the H^1 -norm. In [2] λ is a user-supplied parameter which is shown to be problem-dependent; i.e., it is sensitive to the approximate solution; however, in [3], Adjerid and Flaherty gave a formula to choose the value of λ at each time step, which is shown to be very effective in the numerical experiments. We see that if $\|E_i\|_1$ is larger than $\|\bar{E}\|_1$, then the points $x_i(t)$ and $x_{i-1}(t)$ move together. Similarly, when $\|E_i\|_1$ is smaller than $\|\bar{E}\|_1$, the points $x_i(t)$ and $x_{i-1}(t)$ move apart.

In more recent work, Huang, Ren, and Russell [39, 40] have derived some moving mesh methods based on the EP. In contrast to the above approaches, finite difference

methods are used. First they employ a coordinate transformation $(t, x) \rightarrow (t, \xi)$ with

$$\xi = \frac{1}{\theta} \int_0^x M(\bar{x}) d\bar{x}, \quad (2.10)$$

where $M = \sqrt{1 + u_x^2}$, is the arclength function, and $\theta = \int_0^1 M(\bar{x}) d\bar{x}$. If we think of x as a function of ξ and t , i.e., $x(\xi, t)$, then at the time t , an equidistributed mesh in x is obtained for $\xi_i = i/N$, $i = 0, \dots, N$, i.e., $\{x(i/N, t)\}_{i=0}^N$ is an equidistributed mesh.

If (2.10) is differentiated with respect to ξ twice, a differential form of the EP is obtained,

$$\frac{\partial}{\partial \xi} \left\{ M(x(\xi, t), t) \frac{\partial}{\partial \xi} x(\xi, t) \right\} = 0.$$

Huang et al. require the mesh to satisfy the EP at the later time $t + \tau$ ($0 < \tau \ll 1$), instead of at t . That is, the mesh satisfies

$$\frac{\partial}{\partial \xi} \left\{ M(x(\xi, t + \tau), t + \tau) \frac{\partial}{\partial \xi} x(\xi, t + \tau) \right\} = 0. \quad (2.11)$$

By using Taylor series expansions of $\frac{\partial}{\partial \xi} x(\xi, t + \tau)$ and $M(x(\xi, t + \tau), t + \tau)$ at $x(\xi, t)$

$$\frac{\partial}{\partial \xi} x(\xi, t + \tau) = \frac{\partial}{\partial \xi} x(\xi, t) + \tau \frac{\partial}{\partial \xi} \dot{x}(\xi, t) + O(\tau^2),$$

$$M(x(\xi, t + \tau), t + \tau) = M(x(\xi, t), t) + \tau \dot{x} \frac{\partial}{\partial x} M(x(\xi, t), t) + \tau \frac{\partial}{\partial t} M(x(\xi, t), t) + O(\tau^2).$$

Substituting in (2.11) and keeping only the linear terms, they obtained a moving mesh PDE (MMPDE) of the form

$$\frac{\partial}{\partial \xi} \left(M \frac{\partial \dot{x}}{\partial \xi} \right) + \frac{\partial}{\partial \xi} \left(\frac{\partial M}{\partial \xi} \dot{x} \right) = - \frac{\partial}{\partial \xi} \left(\frac{\partial M}{\partial t} \frac{\partial x}{\partial \xi} \right) - \frac{1}{\tau} \frac{\partial}{\partial \xi} \left(M \frac{\partial x}{\partial \xi} \right). \quad (2.12)$$

By means of central finite differences, the discretization of (2.12) is given on the uniform computational mesh $\xi_i = \frac{i}{N}$, $i = 0, \dots, N$.

Several MMPDEs are derived in [39, 40]. The discrete mesh equations are coupled with the ODEs which come from the discretization of the physical PDEs. DASSL is employed to solve the ODE system. In [41], Huang and Russell presented a moving

collocation method based on those MMPDEs. They used a cubic Hermite collocation method to discretize the physical PDEs and a central difference method to discretize the MMPDEs. Surprisingly, their computational results indicate third-order convergence, which is slower than the traditional (fourth-order) cubic collocation on a fixed mesh but faster than the first-order convergence of the moving finite difference methods in [39, 40]. However, no theoretical analysis to confirm these results appears to have been done.

In summary, the locations of the mesh points and the solutions at these points are obtained simultaneously. Interpolation of the data from the old mesh to the new mesh is not necessary, here, in contrast with some other approaches. Unfortunately, the mesh points can cross or move out of the domain. That is, the physical restrictions, $x_i(t) < x_{i+1}(t)$ and $0 \leq x_i(t) \leq 1$, sometimes do not hold throughout the computation, and some technique to maintain these is necessary. For example, Huang and Russell [42] proved that, by using a spatial smoothing technique, the physical restrictions can be preserved. However, such a technique can increase the complexity of the mesh equations. The most serious restriction in the r-refinement approach is that the number of mesh points is fixed, which makes it impossible to fully control the spatial error. This limitation also makes the r-refinement approach inefficient in the case when a shock evolves into a smooth solution.

2.3 h- and hp-refinement

In the h- and hp-refinement approach, the mesh is refined or coarsened after one or several time steps or at fixed times, and the mesh selection is not coupled with the solution computation. Interpolation after the remeshing process is necessary to generate the initial values for the next step. The hp-refinement approach also allows for a change in the order of accuracy of the spatial discretization, globally or locally.

Sanz-Serna and Christie [64] applied central finite differences to approximate the

spatial derivatives. After each time step t_n , the arclength $\theta(t_n)$ is approximated by a piecewise linear function, i.e. one connects the adjacent mesh points with a straight line. This gives

$$\theta(t_n) \approx \sum_{i=1}^N \sqrt{(x_i^n - x_{i-1}^n)^2 + (U_i^n - U_{i-1}^n)^2}.$$

They required the new mesh points, $\{x_i^*\}_{i=1}^N$, to satisfy

$$\int_{x_{i-1}^*}^{x_i^*} \sqrt{1 + (U_x)^2} dx = \text{constant} = \frac{1}{N} \theta(t),$$

where $i = 1, \dots, N$. After remeshing, the values of the new solutions $\{U_i^*\}_{i=0}^N$ at the new mesh points are obtained from $\{U_i\}_{i=0}^N$ by means of cubic interpolation. We note that since this strategy requires remeshing at each time step, it is not suitable for use with multi-step methods because of the lack of necessary data from previous steps. The authors employed the implicit Euler method for the time integration.

Berzins et al. extended the above idea, which led to the solver called SPRINT [12, 13] and its modified versions in the NAG library, D03PPF and D03PRF. In this code, the user can specify the monitor function, $M(t)$, which is recommended to be the second spatial derivative of the solution approximation since the spatial discretization is done by a second order finite difference method. After a fixed time interval or a fixed number of time steps, specified by the user, a new mesh, $\{x_i^*\}_{i=1}^N$, is calculated by requiring

$$\int_{x_{i-1}^*}^{x_i^*} M(t) dx = \text{constant}.$$

A remeshing is performed only if there exists i , such that the distance between the new position, x_i^* , and the old one, x_i , is more than some fraction of the old mesh spacing on either side of x_i , i.e.,

$$x_i^* > x_i + c(x_{i+1} - x_i),$$

or

$$x_i^* < x_i - c(x_i - x_{i-1}),$$

where $c > 0$ is a user-supplied fraction factor.

These codes use a modification of DASSL as the time integrator. A Theta method [14, 63] (switching between Newton and functional iteration) is implemented in addition to the BDF methods included in DASSL. For an ODE of the form $y' = f(t, y)$, let y_n be the approximate solution at the time t_n , and let y'_n be the first derivative of the approximate solution at the time t_n . At the time t_{n+1} , the Theta method calculates the approximate solution, y_{n+1} , using $y_{n+1} = y_n + \theta y'_n + (1 - \theta)f(t, y_{n+1})$, where θ is chosen by the code at each step. The Theta method has, generally speaking, only first-order accuracy but may be more efficient for large systems of ODEs or DAEs and relatively large tolerances in MOL approaches [15]. A scaling technique, with theoretical justification provided by Petzold and Lötstedt [62], is implemented in the Newton iteration for both the BDF methods and the Theta method. Section 3.2.7 gives a detailed description of this technique.

After remeshing, cubic spline interpolation is used to provide solution values at the current step; interpolation is also used to obtain solution information from previous steps. This makes it possible to use a multi-step scheme. The time integrator then attempts to continue using the stepsize and order determined in the last step prior to remeshing. The Jacobian matrix is recalculated after each remeshing.

Adjerid, Flaherty and Wang [4, 5] use a piecewise polynomial of degree p for the approximate solution $U(x, t)$, where C^0 continuity is required at the mesh points. A Galerkin method is applied to

$$U_t(x, t) = f(t, x, U(x, t), U_x(x, t), U_{xx}(x, t)).$$

An a posteriori error estimate, $E(x, t)$, is obtained as the solution of either a local parabolic or local elliptic finite element problem using piecewise polynomials of degree $p + 1$. First, they calculate $E_i(x, t)$ by solving

$$(U_i)_t + (E_i)_t = f(t, x, U_i + E_i, (U_i)_x + (E_i)_x, (U_i)_{xx} + (E_i)_{xx}),$$

where U_i is the approximate solution. $E_i(x, t)$ is an error estimate on $[x_{i-1}, x_i]$, $i = 1, \dots, N$, which satisfies $E_i(x_{i-1}, t) = E_i(x_i, t) = 0$. Dropping the term, $(E_i)_t$, they obtain another estimate by solving the elliptic equation

$$(U_i)_t = f(t, x, U_i + E_i, (U_i)_x + (E_i)_x, (U_i)_{xx} + (E_i)_{xx}),$$

for $E_i(x, t)$. The H^1 -norm error estimate, $E(x, t)$, is then obtained from

$$\|E(x, t)\|_1^2 = \sum_{i=1}^N \|E_i(x, t)\|_1^2,$$

where

$$\|E_i(x, t)\|_1 = \sqrt{\int_{x_{i-1}}^{x_i} (E_i^2(x, t) + (E_i)_x^2(x)) dx}.$$

Adjerid et al. [5] proved that, for linear Dirichlet problems and using either the parabolic or elliptic error estimate, $E(x, t)$ converged to the exact error in the H^1 -norm when $h \rightarrow 0$, where $h = \max_{1 \leq i \leq N} (x_i - x_{i-1})$. Moore [57] proved that this was also true for the nonlinear case.

Based on the above *a posteriori* error estimate, Adjerid et al. [4] developed PDE-FRONT and Moore [58] extended this work and developed HPDASSL and HPSIRK. All three codes employed hp-refinement. The h-refinement part of the work of these authors can be summarized as follows. After a fixed number of time steps (one step if a Runge-Kutta method is used and more than one if DASSL is applied), the error estimate is computed and an adaptive strategy is invoked when the error is too small or larger than the tolerance. Adjacent elements are combined if they have sufficiently small error. Individual elements are uniformly subdivided if the error is too large.

If the solutions are smooth, high-order approximations can be more efficient than low-order approximations. Therefore, if the order of the piecewise polynomial solution on each subinterval is allowed to vary, the solution can be calculated more efficiently. Guo and Babuska [32, 33, 34] successfully employed this idea, called p-refinement, to elliptic problems. Adjerid et al. [4] combine p-refinement with h-refinement.

Their hp-refinement strategy includes: increasing the polynomial degree in smooth high-error regions, decreasing the polynomial degree in non-smooth low-error regions, coarsening the mesh in smooth low-error regions, and refining the mesh in non-smooth high-error regions. After a remeshing, interpolation is applied to obtain solution values not only for the current step but also for previous steps. For multistep methods this makes it possible to continue integration with a new mesh using the previous stepsize and order. We note that the interpolation is done in a fairly straightforward way since finite elements are used instead of finite differences. Further simplifications in the algorithm are obtained if a Runge-Kutta scheme is used.

The interpolation of data from the old mesh to the new mesh is unavoidable for h- or hp-refinement. Finite element methods provide a more natural interpolation process than finite difference methods. However, if the interpolation error is not small enough, the residual in the current step and the previous steps can change significantly. If the ODE or DAE solver continues the integration, it may be unable to succeed with the same stepsize and order. In some cases, there can even be a complete failure of the time integration. An example of this kind of difficulty arises in Berzins et al. [10], where a second-order finite difference method which is employed for the spatial discretization, is coupled with a cubic spline interpolant or a linear interpolant. They show that if the number of mesh points is not sufficiently large then after a remeshing the interpolation error can pollute the residual and the ODE or DAE solver will not be able to continue the integration with the same stepsize and order. The advantages of h- or hp-refinement are obvious since they allow the number of mesh points to be changed, making it possible to control the spatial error.

2.4 Balancing the Spatial and Time Errors

In 1991, Lawson et al. [48] presented an error control strategy for the time integration of the solution of a system of parabolic equations using the MOL. The authors

restricted their analysis to a numerical scheme with a fixed spatial mesh and a second-order finite difference method for the spatial discretization. However, their strategy has the potential to be combined with spatially adaptive methods.

The strategy is to balance the spatial and time integration errors so that they are of the same order of magnitude. It is clearly most efficient to integrate the resulting ODE or DAE system with just sufficient accuracy so that the temporal error is not significantly more or less than the spatial error. In most existing ODE or DAE software, the standard approach is to control the temporal error with a user-supplied, fixed tolerance. Therefore, in most MOL codes the time tolerance stays unchanged throughout the time integration, which is undesirable since it may provide too much accuracy, compared with the spatial error, in some cases, and not enough in other cases. Since the spatial error varies with time, Lawson et al. adjust the tolerance in the time integration according to the size of the spatial error estimate in each step. This approach suggests how one can develop a fully automatic general-purpose algorithm for fixed-mesh solvers.

Lawson et al. suggest an interesting research direction, in which their idea of balancing the spatial and time errors for a fixed spatial mesh is generalized to incorporate adaptive spatial meshes. For a fixed number of mesh points, r-refinement schemes attempt to move the mesh points to follow the sharp variation of the solution, thus making the best use of the current number of mesh points. If this approach is combined with the above strategy, the user would not need to supply a tolerance; in other words, the MOL code would automatically make the best use of the given number of mesh points while balancing spatial error and the time error. However, to our knowledge, there has been little work done in the past decade on this approach.

2.5 Other Adaptive Packages

In the last decade, several adaptive MOL solvers for 1-D parabolic systems have been developed.

- MOVCOL was developed by Huang and Russell [41]. It is an r-refinement collocation solver, in which the arclength is employed as the monitor function. The mesh equation comes from the differential form of the equidistribution principle (See Section 2.2 for details). For the spatial discretization, a cubic Hermite collocation approach is used for the physical PDE, and a standard finite difference discretization is used for the mesh equations. DASSL is employed to solve the resulting DAE system.
- TOMS731 is an r-refinement package written by Blom and Zegeling [19]. A finite element method of second order accuracy is implemented for the spatial discretization. The time integration is done by DASSL. The monitor function is chosen to be $M = \sqrt{\alpha + \|u_x\|_2^2}$, where α ($\alpha > 0$) is a user-supplied parameter, and $\|\cdot\|_2$ is the L^2 -norm. The parameter α is added to the monitor function to ensure that M is strictly positive. For example, the choice $\alpha = 1$ will result in the arclength monitor.
- D03PPF from the NAG library is a modification of SPRINT by Berzins et al. It employs an h-refinement approach using a fixed number of mesh points (which implies that the code does not control the spatial error). Second-order finite differences are used for the spatial discretization. The user is asked to supply the monitor function and the number of time steps after which a remeshing is performed. A more detailed discussion is given in Chapter 5.
- HPDASSL is a package by Moore [58], based on the hp-refinement approach. The spatial discretization is accomplished using a Galerkin method with a piecewise polynomial basis of degree $p \geq 1$. The spatial error is estimated using a piecewise polynomial of degree $p + 1$. The time integrator is DASSL. After five

time steps, a spatial error estimate is computed and a new mesh (the number of mesh points is allowed to vary) is obtained. However the code does not recompute these five steps if the spatial error violates the spatial tolerance. The tolerance for time integration is set to be 0.1 times the tolerance for spatial discretization. A modification of HPDASSL, HPNEW [59], was recently developed. HPNEW modifies the algorithm which estimates the spatial error, and the tolerance for time integration is set to be 0.02 times the tolerance for spatial discretization. Thus little attention is directed toward balancing the spatial and time error in this code.

- HPSIRK is also an hp-refinement code written by Moore [58]. There are three main differences between HPSIRK and HPDASSL. First, a singly implicit Runge-Kutta (SIRK) method is used for the time integration instead of the BDF method. Second, the spatial error estimate is calculated after every step. The last difference is the most important one: remeshing is allowed at each time step and steps for which the spatial error violates the spatial tolerance are recomputed. Therefore, both the spatial and time errors are controlled. In [58], numerical results show that HPSIRK is more stable but generally much slower than HPDASSL. This is because they apply the Runge-Kutta method instead of the BDF method on the time integration.

Chapter 3

Overview of the BACOL Algorithm

This chapter will describe the new software package, BACOL, for the numerical solution of one dimensional parabolic PDEs, using the h-refinement approach. Collocation with a B-spline basis is applied for the spatial discretization. The flexibility of the B-spline basis software of De Boor [21], upon which BACOL is based, allows us to provide collocation based on piecewise polynomials of arbitrary degree with the restriction that the degree is in the range [3, 11]. The approximate solution is calculated in a piecewise polynomial subspace of degree p , and the spatial error estimate is obtained from a comparison with a second solution computed in a degree $p+1$ piecewise polynomial subspace. Section 3.1 introduces the spatial discretization, including an introduction to the B-spline basis, collocation at Gaussian points, and the treatment of the boundary conditions as algebraic constraints. Section 3.2 describes the time integration component of BACOL, which is a modification of DASSL. This section also describes how to obtain consistent initial conditions for DASSL, the structure of the DAE system arising from the MOL, the addition of a specialized linear system solver to DASSL, and modifications to reduce the condition number of the Newton iteration matrix arising in the treatment of the nonlinear systems. We describe our adaptive strategy in Section 3.3, including an *a posteriori* spatial error estimate, global mesh refinement, and a strategy for predicting the number of spatial subintervals.

3.1 Spatial Discretization

In this section, we first describe the piecewise polynomial subspace, a B-spline basis, which is used in BACOL. We then consider the application of the collocation method at internal Gaussian points using the B-spline basis. The discretization of the PDE system together with the treatment of the boundary conditions in their original form leads to a DAE system.

3.1.1 B-spline Basis

We will consider a mesh consisting of an increasing sequence of $N + 1$ points ($N > 1$) in $[0, 1]$ such that

$$0 = x_0 < x_1 < \cdots < x_N = 1. \quad (3.1)$$

Associated with the mesh are piecewise polynomials of degree p , i.e., there is a polynomial $P_i(x)$ of degree p for each subinterval, $[x_{i-1}, x_i]$, $i = 1, \dots, N$. (Although there could be polynomials of different degrees on different subintervals, we will only discuss the above piecewise polynomial subspace in this thesis.) Piecewise polynomial spaces are popular in MOL codes, especially for finite element or collocation methods. In most MOL codes there are continuity conditions at each internal mesh point, x_i , $i = 1, \dots, N - 1$. For example, EPDCOL [53] asks for C^1 -continuity at internal mesh points while PDECHEB [11] requires only C^0 -continuity. C^1 -continuity is used in BACOL. Consequently the dimension of this piecewise polynomial subspace, S_p , is given by

$$\begin{aligned} \dim(S_p) = NC &= N(p + 1) - 2(N - 1) \\ &= N(p - 1) + 2. \end{aligned}$$

There are several types of basis functions that can be used to represent piecewise polynomials, such as B-splines [53], Chebyshev polynomials [11], monomial splines [47], and Hermite polynomials [41]. For boundary value ODE problems, the monomial spline basis has some advantages over B-splines and other splines (See, e.g., [8]). We

first tried the monomial spline basis for the piecewise polynomial subspace (before we ultimately turned to a B-spline basis). However, some of the advantages of monomial splines do not hold any more (some of them even become a disadvantage) due to the difference between PDEs and ODEs. First, in the ODE case, after the spatial discretization an algebraic equation system is generated. In the Newton iteration process, a condensation technique can be applied to the Jacobian matrix [8]. This reduces the size of the Jacobian matrix to the same size as the Jacobian matrix using a B-spline basis. However, in the PDE case, after the spatial discretization an index-1 DAE system is generated (the definition of index and the further discussion for index-1 DAE systems will be described in Section 3.2.1). This makes it impossible to carry out a condensation technique. Therefore, using a monomial spline basis leads to a larger DAE system than using a B-spline basis; i.e., more computational effort is needed. Second, in the ODE case, the biggest advantage for monomial splines is that the Jacobian matrices have a much smaller condition number (the definition of condition number will be described in Section 3.2.7) than the condition number of the Jacobian matrices using other splines [8]. However, this does not hold for the PDE cases. In fact, the condition number of the Jacobian matrices using a monomial spline basis is even larger than the condition number of the Jacobian matrices using a B-spline basis. The reason is, after the spatial discretization, using a monomial spline basis will lead to an index-1 DAE system while using a B-spline basis and differentiating the boundary conditions (as EPDCOL [53] does) will lead to a pure ODE system; i.e., an index-0 DAE. Thus, by using a monomial spline basis, the Jacobian matrices have a condition number of order $O(1/h)$, where h is the stepsize of the time integration; however, by using a B-spline basis, the condition number is only of order $O(1)$ (the relationship between the index of DAE and the condition number of Jacobian can be found in Section 3.2.7). During our early work, we found that, when a small tolerance is used (e.g., 10^{-8}), MSCPDE [47, 60], an MOL package which uses a monomial spline basis and uses a modification version of DASSL as the time integrator, occasionally was not able to start the time integration at the first

step. This is because DASSL tends to try an initial stepsize which is close to the size of the tolerance. So at the first step, the convergence test or the error test may repeatedly fail because of the large condition number. Based on the above reasons, we decided against monomial splines. Our early approach was to use a B-spline basis and differentiate the boundary conditions (BCs). However we found that differentiating the BCs led to other difficulties (See Section 5.4 for details). Therefore we apply the BCs directly, which of course leads to an index-1 DAE system again. A scaling technique (which is described in Section 3.2.7) is applied to reduce the condition number.

Because of the above reasons, in the BACOL code, we employ the B-spline package for the spatial discretization. The well-known B-spline package [21, 22] allows for a stable representation of piecewise polynomials of arbitrary degree and arbitrary continuity. For more details regarding B-splines, we refer the reader to [22].

We recall that the dimension of S_p is NC and we will denote $\{B_i(x)\}_{i=1}^{NC}$ as the corresponding B-spline basis. This basis has the following well-known property, (See e.g. [21])

for any x such that $x_{k-1} < x < x_k$, $1 \leq k \leq N$, at most $p + 1$ B-spline basis functions, namely, $\{B_i(x)\}_{i=(k-1)(p-1)+1}^{k(p-1)+2}$, have nonzero values.

The special structure of the Newton iteration matrix associated with the time integration (which will be discussed in Section 3.2.3) is derived on the basis of this property.

3.1.2 Collocation at Gaussian Points

For convenience, we will consider (1.1)-(1.4) with $t_0 = 0$, $x_a = 0$, and $x_b = 1$; then the general form becomes

$$u_t(x, t) = f(t, x, u(x, t), u_x(x, t), u_{xx}(x, t)), \quad 0 \leq x \leq 1, \quad t \geq 0, \quad (3.2)$$

$$u(x, 0) = u_0(x), \quad 0 \leq x \leq 1, \quad (3.3)$$

$$b_L(t, u(0, t), u_x(0, t)) = 0, \quad t \geq 0, \quad (3.4)$$

$$b_R(t, u(1, t), u_x(1, t)) = 0, \quad t \geq 0. \quad (3.5)$$

We will assume that a mesh of the form (3.1), along with the B-spline basis, $\{B_i(x)\}_{i=1}^{NC}$, is given. The approximate solution $U(x, t)$ is then expressed as a linear combination of B-spline basis functions in space, with time-dependent coefficients to be determined. The s -th component of the solution, $u_s(x, t)$, of (3.2)-(3.5) is then approximated by a piecewise polynomial, $U_s(x, t)$, of degree p in x , which is of the form

$$U_s(x, t) = \sum_{i=1}^{NC} B_i(x) y_{i,s}(t). \quad (3.6)$$

We then collocate at $p - 1$ collocation points in each subinterval, i.e. we require the approximate solution to satisfy the PDEs at a certain set of the collocation points, described next.

Gauss-Legendre points in each subinterval are used as the collocation points in several codes, e.g. PDECOL [53], COLSYS [6], and MSCPDE [47]. De Boor and Swartz [23] have proved that, when collocating at Gaussian points for boundary value problems, the error at mesh points is particularly small compared to the error at other locations. This high-order convergence at the mesh points is called *superconvergence*. Although, to our knowledge, there are no similar results for the PDE case, we will choose Gauss-Legendre points to be the collocation points. (In Section 5.6, we will experimentally demonstrate the presence of superconvergence for the PDE case.)

Now we describe, in detail, the collocation method which provides our spatial discretization. First, define the mesh step size sequence $\{h_i\}_{i=1}^N$ by $h_i = x_i - x_{i-1}$. Then let $\{\rho_i\}_{i=1}^{p-1}$ be the Canonical Gaussian points on $[0, 1]$ such that

$$0 < \rho_0 < \rho_1 < \cdots < \rho_{p-1} < 1.$$

The collocation points are then defined by

$$\xi_1 = 0, \quad (3.7)$$

$$\begin{aligned} \xi_l &= x_{i-1} + h_i \rho_j, \quad \text{where } l = 1 + (i-1) * (p-1) + j, \\ &\text{for } i = 1, \dots, N, \quad j = 1, \dots, p-1, \end{aligned} \quad (3.8)$$

$$\xi_{NC} = 1, \quad (3.9)$$

where $NC = N(p-1) + 2$ is the number of collocation points. We note the collocation point sequence, $\{\xi_l\}_{l=1}^{NC}$, includes the internal (Gaussian) points and the two boundary points.

We require the approximate solution, $U_s(x, t)$, to satisfy the original PDEs at the internal collocation points; we thus obtain

$$\frac{d}{dt} U_s(\xi_l, t) = f_s(t, \xi_l, U(\xi_l, t), U_x(\xi_l, t), U_{xx}(\xi_l, t)), \quad (3.10)$$

where $l = 2, \dots, NC - 1$ and $s = 1, \dots, NPDE$. This results in $N(p-1)$ collocation equations for each component.

We note that at each internal collocation point, there are at most $p+1$ nonzero B-spline basis functions which have nonzero values. We therefore obtain

$$U_s(\xi_l, t) = \sum_{m=(i-1)(p-1)+1}^{i(p-1)+2} B_m(\xi_l) y_{m,s}(t), \quad (3.11)$$

where $\xi_l = x_{i-1} + h_j \rho_j$ is the j -th Gaussian point of the i -th subinterval and $s = 1, \dots, NPDE$. Substituting (3.11) into (3.10), we obtain

$$\sum_{m=(i-1)(p-1)+1}^{i(p-1)+2} B_m(\xi_l) y'_{m,s}(t) = f_s(t, \xi_l, U(\xi_l, t), U_x(\xi_l, t), U_{xx}(\xi_l, t)), \quad (3.12)$$

where $l = 1, \dots, NC$, $s = 1, \dots, NPDE$. We note that the function, $f_s(t, \xi_l, U(\xi_l, t), U_x(\xi_l, t), U_{xx}(\xi_l, t))$, depends only on $\{y_{m,s}(t)\}$, $m = (i-1)(p-1) + 1, \dots, i(p-1) + 2$ and $s = 1, \dots, NPDE$.

3.1.3 Treatment of the Boundary Conditions

In Chapter 1, we described several ways to treat the boundary conditions (BCs). In this thesis, we treat the boundary conditions in their original form,

$$b_L(t, U(0, t), U_x(0, t)) = 0, \quad (3.13)$$

$$b_R(t, U(1, t), U_x(1, t)) = 0. \quad (3.14)$$

We note that due to special properties of the B-spline basis (See [21])

$$U_s(0, t) = B_1(0)y_{1,s}(t), \quad (3.15)$$

$$(U_s)_x(0, t) = B'_1(0)y_{1,s}(t) + B'_2(0)y_{2,s}(t), \quad (3.16)$$

$$U_s(1, t) = B_{NC}(1)y_{NC,s}(t), \quad (3.17)$$

$$(U_s)_x(1, t) = B'_{NC}(1)y_{NC,s}(t) + B'_{NC-1}(1)y_{NC-1,s}(t), \quad (3.18)$$

where $s = 1, \dots, NPDE$. Therefore, only B_1 , B'_1 , and B'_2 appear in the equations associated with the left boundary, and only B_{NC} , B'_{NC} , and B'_{NC-1} appear in the ones associated with the right boundary.

We see that, after the spatial discretization, (i.e. after the application of the collocation conditions,) the resultant ODEs coupled with the boundary conditions give a differential-algebraic equation system

$$b_L(t, U(0, t), U_x(0, t)) = 0, \quad (3.19)$$

$$\sum_{m=(i-1)(p-1)+1}^{i(p-1)+2} B_m(\xi_l)y'_{m,s}(t) = f_s(t, \xi_l, U(\xi_l, t), U_x(\xi_l, t), U_{xx}(\xi_l, t)), \quad (3.20)$$

where ξ_l is one of the $p - 1$ collocation points in the i -th subinterval, $i = 1, \dots, N$, $s = 1, \dots, NPDE$,

$$b_R(t, U(1, t), U_x(1, t)) = 0. \quad (3.21)$$

In total there are $N(p - 1) + 2$ equations for each component. Thus we generate the same number of equations as the number of unknowns.

Since we will later compare the effect of different ways of treating the BCs as considered in BACOL and EPDCOL [46], we will now also explain how EPDCOL treats the BCs. EPDCOL discretizes the PDEs using the same collocation scheme described above. However, instead of keeping the BCs, (3.13), as they are, EPDCOL expects them to be differentiated with respect to time, giving, at $x = 0$,

$$\begin{aligned} \sum_{j=1}^{NPDE} \left(\frac{\partial(b_L)_s}{\partial U_j} B_1(0) + \frac{\partial(b_L)_s}{\partial(U_x)_j} B_1'(0) \right) \frac{dy_{1,j}(t)}{dt} \\ + \sum_{j=1}^{NPDE} \left(\frac{\partial(b_L)_s}{\partial(U_x)_j} B_2'(0) \right) \frac{dy_{2,j}(t)}{dt} = -\frac{\partial(b_L)_s}{\partial t}, \end{aligned} \quad (3.22)$$

where $s = 1, \dots, NPDE$. Similarly at $x = 1$ we have

$$\begin{aligned} \sum_{j=1}^{NPDE} \left(\frac{\partial(b_R)_s}{\partial U_j} B_{NC}(1) + \frac{\partial(b_R)_s}{\partial(U_x)_j} B'_{NC}(1) \right) \frac{dy_{NC,j}(t)}{dt} \\ + \sum_{j=1}^{NPDE} \left(\frac{\partial(b_R)_s}{\partial(U_x)_j} B'_{NC-1}(1) \right) \frac{dy_{NC-1,j}(t)}{dt} = -\frac{\partial(b_R)_s}{\partial t}, \end{aligned} \quad (3.23)$$

where $s = 1, \dots, NPDE$. We note that EPDCOL thus obtains an ODE system after its spatial discretization by including the differentiated BCs (3.22), (3.23), rather than the original BCs in algebraic form.

3.2 Time Integration

A sophisticated ODE or DAE integrator is essential to the success of a MOL code. Since a major computational effort is associated with the linear systems which arise in the calculations performed by the ODE or DAE solver, it is important to employ a suitable linear algebra package which takes advantage of any special structure possessed by the linear system. In this section, we first discuss some properties of DAEs. Then some special structure of the Newton iteration matrices, which arises due to the properties of the B-splines, is identified. We next discuss a linear system solver,

COLROW [30], which takes advantage of this special structure. Finally we discuss a modification to DASSL, which involves adding a new linear system option to include COLROW.

3.2.1 Index of DAEs

A property known as the *index* plays an important role in the classification and the numerical solution of DAE's. Before we give the definition of the index, we consider a special case of DAEs in Hessenberg form

$$y' = F_1(t, y, z), \quad (3.24)$$

$$0 = F_2(t, y, z), \quad (3.25)$$

where y and z are vectors. If we differentiate the algebraic constraint equation (3.25) with respect to t , we obtain

$$y' = F_1(t, y, z), \quad (3.26)$$

$$\frac{\partial F_2}{\partial y} y' + \frac{\partial F_2}{\partial z} z' = -\frac{\partial F_2}{\partial t}. \quad (3.27)$$

If $\frac{\partial F_2}{\partial z}$ is nonsingular, the system (3.26), (3.27) may be rewritten as a standard ODE system for y' and z' , and we say that (3.24), (3.25) has index one. If this is not the case, there exist two nonsingular matrices, L and U , such that

$$L \frac{\partial F_2}{\partial z} U = \begin{pmatrix} I_m & 0 \\ 0 & 0 \end{pmatrix}, \quad (3.28)$$

where I_m is an m -by- m unit matrix for some m . Therefore, using the coordinate transformation $\hat{z} = U^{-1}z$ and (3.28), we are able to write (3.27) as

$$\begin{pmatrix} I_m & 0 \\ 0 & 0 \end{pmatrix} \hat{z}' = -L \frac{\partial F_2}{\partial y} y' - L \frac{\partial F_2}{\partial t}. \quad (3.29)$$

We thus are able to write (3.26), (3.29), in the form of (3.24), (3.25), but with different unknown solution components. We then differentiate the algebraic equations again

with respect to t . If we obtain an ODE system, we say that the original problem has index two. However, if again this is not the case, we repeat the above process. The number of differentiation steps required in this procedure is the differential index. We have the following definition for the general case, as given in [24]

Definition 1 *The minimum number of times that all or part of $F(t, y, y') = 0$ must be differentiated with respect to t in order to determine $y'(t)$ as a continuous function of $y(t)$ and t is the index of the DAE.*

From the above definition of the index, we note that the DAE system (3.19)-(3.21) obtained from our spatial discretization has index one since differentiating the boundary conditions leads to a system of ODEs of the form

$$M(t, y)y' = G(t, y),$$

where $M(t, y)$ is non-singular (See [53]).

3.2.2 A Short Description of BDF Methods for DAEs

The backward differential formulas (BDF) are the most popular linear multistep methods for DAEs. DASSL [61], the most widely used DAE code, is based on BDF methods. In BACOL, the time integration is performed using a modification of DASSL. Here we give a brief description of BDF methods (for further information, See, e.g. [24]).

BDF methods can be used to solve DAEs of the general form

$$F(t, y, y') = 0. \tag{3.30}$$

Let y_n denote the approximate solution at t_n and let y_{n+1} denote the approximate solution at t_{n+1} . The simplest BDF method, which is of the first order, is the implicit Euler method. It replaces $y'(t)$ in (3.30) at t_{n+1} by a backward difference

$$F\left(t_{n+1}, y_{n+1}, \frac{y_{n+1} - y_n}{h_{n+1}}\right) = 0,$$

where $h_{n+1} = t_{n+1} - t_n$. The resulting nonlinear algebraic equation system for y_{n+1} is usually solved by a Newton iteration. If a constant stepsize, h , is used, a k -step BDF replaces $y'(t)$ by the derivative of the polynomial at t_{n+1} that interpolates the computed solution at $t_{n+1}, t_n, \dots, t_{n-k+1}$. Thus we obtain

$$F \left(t_{n+1}, y_{n+1}, \frac{\sum_{i=0}^k \alpha_i y_{n-i+1}}{h} \right) = 0,$$

where $\alpha_i, i = 0, 1, \dots, k$ are the coefficients of the BDF method.

There are three standard implementations of BDF methods that allow for time adaptivity, i.e., variable stepsizes. They are called the fixed coefficient (e.g., LSODI [38]), variable coefficient (e.g., EPISODE [27]), and fixed leading coefficient formulas (e.g., DASSL). The fixed coefficient methods are efficient for smooth problems, but “inefficient or possibly unstable for problems which require frequent adjustment of the stepsize” [24]. The variable coefficient methods are the most stable, but they require more Jacobian evaluations, and are therefore less efficient. The fixed leading coefficient approach is a compromise between these two approaches. It is less stable, but at the same time requires fewer Jacobian evaluations than the variable coefficient formulation. Using a fixed leading coefficient implementation, a k -step BDF method has the form

$$F \left(t_{n+1}, y_{n+1}, \frac{\alpha}{h_{n+1}} y_{n+1} + \beta \right) = 0, \quad (3.31)$$

where α and β are known. The unknown solution value, y_{n+1} , is obtained by using a Newton iteration of the form

$$y_{n+1}^{(m+1)} = y_{n+1}^{(m)} - G^{-1} F \left(t_{n+1}, y_{n+1}^{(m)}, \frac{\alpha}{h_{n+1}} y_{n+1}^{(m)} + \beta \right),$$

where $y_{n+1}^{(m)}$ is the m -th Newton approximation to y_{n+1} and G is an iteration matrix of the form,

$$G = \frac{\alpha}{h_{n+1}} \frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y}. \quad (3.32)$$

We now summarize some useful results about BDF methods for index-1 DAEs:

- A k -step BDF method is unconditionally unstable for $k \geq 7$, and most ODE or DAE codes, e.g., DASSL and LSODI, only implement BDF methods up to order $k = 5$.
- The numerical solution of an index-1 DAE system by a k -step BDF with *fixed* stepsize h for $k < 7$ has a convergence rate of $O(h^k)$ if all initial values are correct to $O(h^k)$ accuracy and if the Newton iteration error is $O(h^{k+1})$.
- For *variable* stepsize BDF methods, if the ratio of stepsize at the adjacent steps is kept bounded, all initial values are correct to $O(h^k)$, and the Newton iteration error is $O(h^{k+1})$, then the error in the numerical solution for a k -step BDF ($k < 7$) applied to a index-1 *constant coefficient* DAE system (i.e., the DAE has the form $Ay' + By = f(t)$, where A and B are constant matrices) is $O(h^k)$.

3.2.3 Structure of the Iteration Matrix

We now return to the index-1 DAE system identified in Section 3.1.3. It has the form

$$b_L(t, U(0, t), U_x(0, t)) = 0, \quad (3.33)$$

$$\sum_{m=(i-1)(p-1)+1}^{i(p-1)+2} B_m(\xi_i) y'_{m,s}(t) - f_s(t, \xi_i, U(\xi_i, t), U_x(\xi_i, t), U_{xx}(\xi_i, t)) = 0, \quad (3.34)$$

where ξ_i is one of the $p - 1$ collocation points in the i -th subinterval, and $s = 1, \dots, NPDE$,

$$b_R(t, U(1, t), U_x(1, t)) = 0. \quad (3.35)$$

We know that

$$\begin{aligned}
 U_s(\xi_l, t) &= \sum_{m=(i-1)(p-1)+1}^{i(p-1)+2} B_m(\xi_l) y_{m,s}(t), \\
 (U_s)_x(\xi_l, t) &= \sum_{m=(i-1)(p-1)+1}^{i(p-1)+2} B'_m(\xi_l) y_{m,s}(t), \\
 (U_s)_{xx}(\xi_l, t) &= \sum_{m=(i-1)(p-1)+1}^{i(p-1)+2} B''_m(\xi_l) y_{m,s}(t),
 \end{aligned}$$

where $\xi_l = x_{i-1} + h_j \rho_j$ is the j -th Gaussian point in the i -th subinterval, and $s = 1, \dots, NPDE$. We also recall (3.15)-(3.18) which gives the dependence of $U_k(0, t)$, $(U_k)_x(0, t)$, $U_k(1, t)$, $(U_k)_x(1, t)$ on $y_{1,k}(t)$, $y_{2,k}(t)$, $y_{NC,k}(t)$, $y_{NC-1,k}(t)$.

From (3.32), we note that the Newton iteration matrix depends on $\frac{\partial F}{\partial y'}$ and $\frac{\partial F}{\partial y}$, and we see that $\frac{\partial F}{\partial y'}$ has the form shown in Figure 3.1

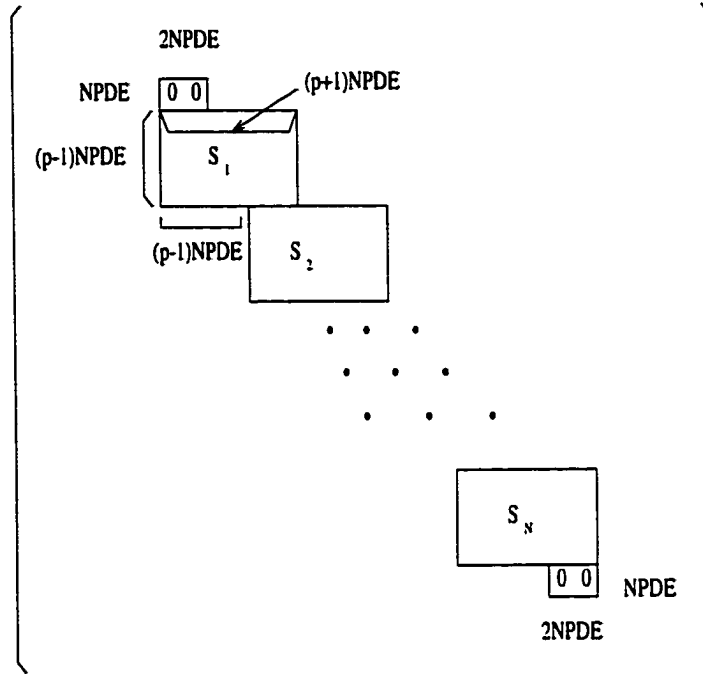


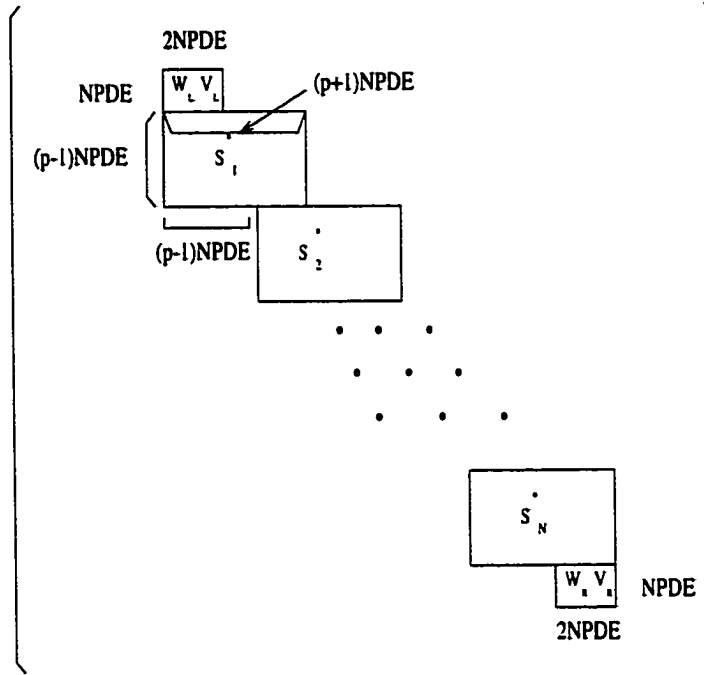
Figure 3.1: Structure of $\frac{\partial F}{\partial y'}$.

Since the boundary conditions are treated as algebraic constraints that have no dependency on y' , the top block and bottom blocks are each $NPDE$ by $2NPDE$ null matrices. Each S_i , $i = 1, \dots, N$, is a $(p-1)NPDE$ by $(p+1)NPDE$ matrix of the form

$$\begin{bmatrix} B_l(\xi_{l+1})I_{NPDE} & B_{l+1}(\xi_{l+1})I_{NPDE} & \cdots & B_{l+p}(\xi_{l+1})I_{NPDE} \\ B_l(\xi_{l+2})I_{NPDE} & B_{l+1}(\xi_{l+2})I_{NPDE} & \cdots & B_{l+p}(\xi_{l+2})I_{NPDE} \\ \vdots & \vdots & \ddots & \vdots \\ B_l(\xi_{l+p-1})I_{NPDE} & B_{l+1}(\xi_{l+p-1})I_{NPDE} & \cdots & B_{l+p}(\xi_{l+p-1})I_{NPDE} \end{bmatrix},$$

where $l = 1 + (i-1)(p-1)$. We note that the matrix structure shown in Figure 3.1 is an almost block diagonal (ABD) form [30].

The matrix, $\frac{\partial F}{\partial y}$, also has the ABD form shown in Figure 3.2

Figure 3.2: Structure of $\frac{\partial F}{\partial y}$.

Here W_L , V_L , W_R and V_R are $NPDE$ by $NPDE$ matrices whose components are of the form

$$(W_L)_{s,j} = \frac{\partial(b_L)_s}{\partial U_j} B_1(0) + \frac{\partial(b_L)_s}{\partial(U_x)_j} B'_1(0), \quad (3.36)$$

$$(V_L)_{s,j} = \frac{\partial(b_L)_s}{\partial(U_x)_j} B'_2(0), \quad (3.37)$$

$$(W_R)_{s,j} = \frac{\partial(b_R)_s}{\partial U_j} B_{NC}(1) + \frac{\partial(b_R)_s}{\partial(U_x)_j} B'_{NC}(1), \quad (3.38)$$

$$(V_R)_{s,j} = \frac{\partial(b_R)_s}{\partial(U_x)_j} B'_{NC-1}(1). \quad (3.39)$$

The i -th subblock, S_i^* , $i = 1, \dots, N$, is an $(p-1)NPDE$ by $(p+1)NPDE$ matrix of the form,

$$\begin{bmatrix} D_{1,1} & D_{1,2} & \cdots & D_{1,p+1} \\ D_{2,1} & D_{2,2} & \cdots & D_{2,p+1} \\ \vdots & \vdots & \ddots & \vdots \\ D_{p-1,1} & D_{p-1,2} & \cdots & D_{p-1,p+1} \end{bmatrix},$$

where each of $D_{m,j}$, $m = 1, \dots, p-1$, $j = 1, \dots, p+1$, is an *NPDE* by *NPDE* matrix, whose components have the form of

$$(D_{m,j})_{s,l} = \frac{\partial f_s}{\partial U_l} B_q(\xi_r) + \frac{\partial f_s}{\partial (U_l)_x} B'_q(\xi_r) + \frac{\partial f_s}{\partial (U_l)_{xx}} B''_q(\xi_r), \quad (3.40)$$

where $q = (i-1)(p-1) + j$, $r = (i-1)(p-1) + 1 + m$, $s = 1, \dots, NPDE$ and $l = 1, \dots, NPDE$.

The Newton iteration matrix, G , is then obtained using (3.32). We note that the Newton iteration matrix also has an ABD form. A linear system solver COLROW [30], which will be introduced shortly, is more efficient in treating the ABD systems than the banded solver which is available in DASSL. Our modification to introduce a new linear algebra solver option within DASSL will be described later. If the user is required to supply $\frac{\partial f}{\partial U}$, $\frac{\partial f}{\partial U_x}$ and $\frac{\partial f}{\partial U_{xx}}$, then the Jacobian matrix can be computed directly; otherwise DASSL provides a finite difference Jacobian option (which is currently not able to take advantage of the ABD form).

For the spatial error control algorithm (described later), we solve the PDEs using first a degree p piecewise polynomial for the spatial discretization, and then a degree $p+1$ piecewise polynomial. These two approximations are compared to estimate the spatial error. The two systems of DAEs (resulting from the spatial discretizations) are not solved independently for reasons explained later, but are instead combined into one large system. This results in a combined Newton iteration matrix having the decoupled form

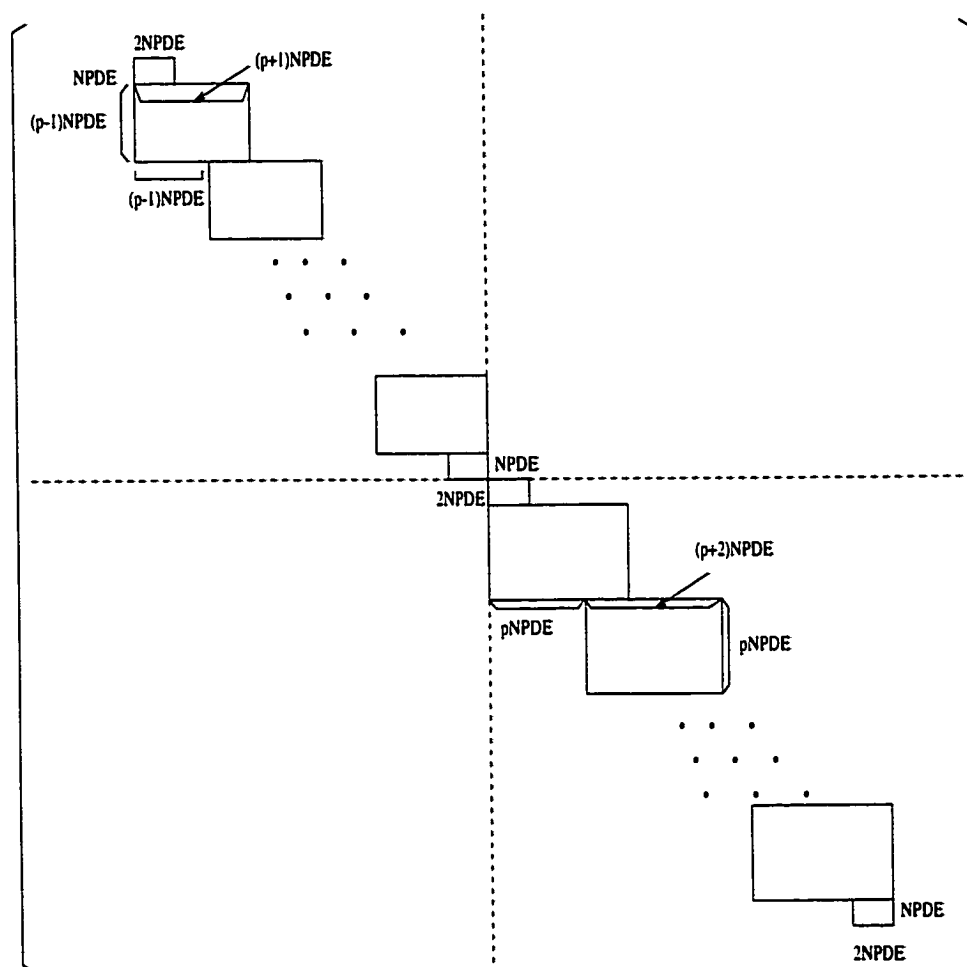


Figure 3.3: Structure of the iteration matrix in BACOL

As a result, we have had to make further changes to DASSL (described later).

3.2.4 Consistent Initial Conditions for the DAE System

The algebraic constraints in a system of DAEs require consistent initial conditions. In fact, small inconsistencies in the initial values may cause DASSL to fail on the first step or to become very inefficient; i.e., a very small stepsize is used for the first few steps after several failed attempted steps.

As discussed in [49], in order to be consistent, the initial conditions of an index-1 DAE must satisfy not only the algebraic constraints, but also the differentiated algebraic constraints. Therefore, we require that our initial conditions satisfy the boundary conditions and, at the internal collocation points ξ_l , $l = 2, \dots, NC - 1$, we require that the piecewise polynomial agrees with the initial conditions of the PDE system; i.e.,

$$\begin{aligned} b_L(t, U(0, 0), U_x(0, 0)) &= 0, \\ U(\xi_l, 0) &= u_0(\xi_l), \quad l = 2, \dots, NC - 1, \\ b_R(t, U(1, 0), U_x(1, 0)) &= 0. \end{aligned}$$

We note that a Newton iteration is needed in order to solve the system of algebraic equations unless Dirichlet boundary conditions are applied on both boundaries. If any of the BCs are not Dirichlet, the initial guess for the Newton iteration is obtained by solving the linear system, $U(\xi_l, 0) = u_0(\xi_l)$, $l = 1, \dots, NC$, i.e., we require that the piecewise polynomial agrees with the initial conditions of the PDEs on both boundaries.

Once we obtain $y(0)$, the value $y'(0)$ is obtained by solving

$$\frac{d}{dt} b_L(t, U(0, 0), U_x(0, 0)) = 0, \quad (3.41)$$

$$U_t(\xi_l, 0) = f(t, \xi_l, U(\xi_l, 0), U_x(\xi_l, 0), U_{xx}(\xi_l, 0)), \quad (3.42)$$

$$l = 2, \dots, NC - 1, \quad (3.43)$$

$$\frac{d}{dt} b_R(t, U(1, 0), U_x(1, 0)) = 0. \quad (3.44)$$

We see that the differentiated boundary conditions are included in the above equations, which ensures consistent initial conditions for DASSL.

3.2.5 Replacement of the Linear Algebra Solver

Based on properties of the B-splines and on the fact that we simultaneously consider the spatial discretizations based on collocation with piecewise polynomials of degree p

and $p + 1$, we were able to show, in Section 3.2.3, that the Jacobian matrix consists of two decoupled almost block diagonal matrices. Consequently we can treat the linear system in a separated way, i.e., by solving the two ABD linear systems independently. Therefore, in this section we describe an efficient way to solve an ABD system which takes full advantage of the structure. Let us look at a simple example with $p = 4$, whose corresponding matrix has the form

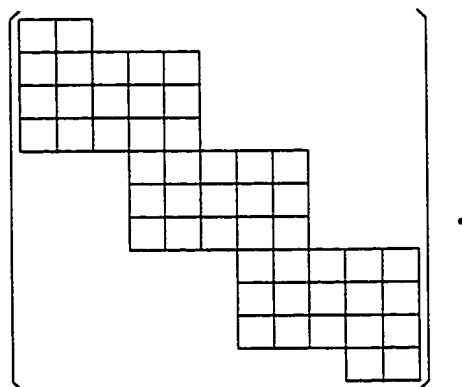


Figure 3.4: Structure of the iteration matrix for $p = 4$.

Each of the small blocks is of size $NPDE$ by $NPDE$. There are two linear system solvers inside DASSL, a full matrix solver and a banded matrix solver. An obvious choice is to treat the above matrix as a banded one. That is, we might consider the form

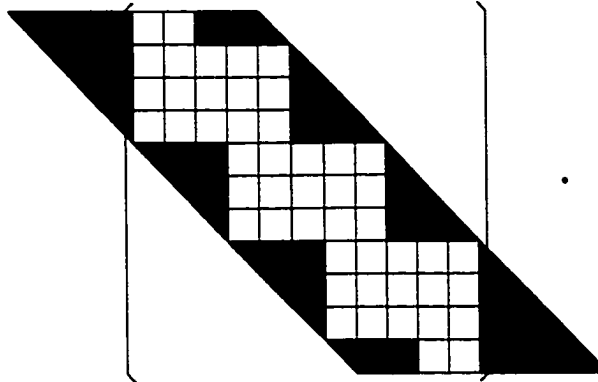


Figure 3.5: Structure of the iteration matrix for $p = 4$.

The shaded area is the extra storage required by the banded solver. We see the total storage for a banded solver is $2 * NPDE^2 * p * (N * (p - 1) + 2)$, while the nonzero part is only $NPDE^2 * (N * (p^2 - 1) + 2)$. When N is large, the total storage is almost twice that of the nonzero part. To take advantage of the ABD structure, Diaz et al. [30] developed the package COLROW, which factors the ABD matrix using an alternating row and column elimination approach which introduces no fill-in. Considering only cubic terms, the total number of multiplications using a banded solver is $2 * N * NPDE^3 * (p + 1)^3$, while the total number of multiplications for COLROW is $\frac{1}{3} * N * NPDE^3 * (p + 1)^3$ (See [46] for details). The cost for COLROW is therefore roughly $\frac{1}{6}$ of that of the banded solver.

We have therefore inserted a new linear system solver option within DASSL to allow the use of COLROW. This has required modification of three subroutines, DDASSL, DDAJAC, and DDASLV. As mentioned before, we solve the decoupled systems separately and COLROW is therefore called twice for each full Newton iteration.

3.2.6 Restarting DASSL after a Remeshing

At certain times during the computation performed by a code that does spatial adaptivity through h-refinement, a new spatial mesh will be computed and it will then

be necessary to continue the time integration starting from this point in time. The simplest approach is called a *cold start*. In this approach, the code interpolates the solution from the old mesh to the new mesh at the current step, and then restarts the integration as though solving a new problem. This, however, is not efficient because the effort to perform a cold start of an ODE or DAE solver is considerable since it asks the solver to perform a tiny stepsize at the beginning of the integration. The second approach is called a *continuation of integration*. As suggested in [10], the code interpolates the solution from the latest step, as well as from as many previous steps as necessary. This means interpolating all the history vectors which are required by the ODE or DAE solver. The same stepsize and order as in the last time step are then tried afterwards. Computational results in [10] support the conclusion that the use of continuation of integration can significantly improve efficiency.

As mentioned before, there are two approximate solutions in BACOL, one based on a piecewise polynomial of degree p and another of degree $p + 1$ respectively. Thus we require the interpolated history vector for both approximations. In BACOL, we obtain both of these interpolates using only the degree $p + 1$ approximation. The reason for doing this is discussed later.

To continue the integration using DASSL, we need to employ interpolation at the current step and at most the last 5 time steps; this is because DASSL implements BDF methods up to a maximum order of 5. From an inspection of DASSL, we see that the divided differences (to be described shortly) for the new mesh are required in order to carry out a continuation of integration.

Let y_n denote the B-spline coefficients at t_n . The (Newton) divided differences are defined by the recursive formulas

$$\begin{aligned} [y_n] &= y_n, \\ [y_n, y_{n-1}, \dots, y_{n-k}] &= \frac{[y_n, y_{n-1}, \dots, y_{n-k+1}] - [y_{n-1}, y_{n-2}, \dots, y_{n-k}]}{t_n - t_{n-k}}. \end{aligned}$$

When a k -th order BDF scheme is employed, DASSL needs the following divided differences

$$[y_n], [y_n, y_{n-1}], [y_n, y_{n-1}, y_{n-2}], \dots, [y_n, y_{n-1}, \dots, y_{n-k}].$$

We see that we can obtain the divided differences for the new mesh if we have the values, y_{n-i} , $i = 0, \dots, k$, over the new mesh. For convenience, we denote by y the B-spline coefficients over the old mesh and by y^* the B-spline coefficients over the new mesh. Let $U^*(x, t)$ denote the approximate solution, NC^* be the number of new collocation points (NC^* is not necessarily the same as NC) and ξ_l^* , $l = 1, \dots, NC^*$, be the collocation points over the new mesh. Similarly let $U(x, t)$ denote the approximate solutions over the old mesh. We then determine $U^*(x, t)$ by requiring it to interpolate U values at the new collocation points, i.e. the following equations are satisfied

$$U^*(\xi_l^*, t_{n-i}) = U(\xi_l^*, t_{n-i}), \quad l = 1, \dots, NC^*, \quad (3.45)$$

where $i = 0, \dots, k$.

3.2.7 Treatment of Poorly Conditioned Jacobian matrices

In this section we first study how the conditioning of the Newton iteration matrix affects our approximate solution. We then describe a technique to improve the accuracy of the solution of the linear system by a transformation of the Jacobian matrix.

The system of linear equations to be solved in each Newton iteration is of the form

$$Ax = b.$$

Let $\|x\|$ be an arbitrary vector norm and $\|A\|$ be the corresponding induced matrix norm. If the solution changes by Δx , corresponding to a change Δb in the right-hand side, we know that the relative change in x is related to the relative change in b by (a proof can be found in any standard numerical analysis text; e.g., [69])

$$\frac{\|\Delta x\|}{\|x\|} \leq \|A\| \|A^{-1}\| \frac{\|\Delta b\|}{\|b\|}.$$

We define the condition number of A , $\text{cond}(A) = \|A\| \|A^{-1}\|$, which is a measure of the sensitivity of the relative error in the solution x to the changes in b . We note that if $\text{cond}(A)$ is large, a small change in b can cause a large change in the solution. We say that A is ill conditioned or poorly conditioned if its condition number is large.

The presence of an ill-conditioned iteration matrix represents a difficulty for numerical schemes for DAE systems. The following theorem is given in [24], page 144,

Theorem 1 *The condition number of the iteration matrix for a DAE system with index γ is $O(h^{-\gamma})$.*

Let us look at the following sample DAE system, which is studied by Petzold and Lötstedt [62]

$$\begin{aligned} F_1(t, y, y', z) &= 0, \\ F_2(t, y, z) &= 0. \end{aligned}$$

Consider attempting to solve the above system to obtain y_{n+1} and z_{n+1} , the solution approximation, at t_{n+1} using a k -step BDF method with the fixed leading coefficient implementation (as DASSL does). We then obtain the system of nonlinear equations

$$F_1(t_{n+1}, y_{n+1}, \frac{\alpha}{h_{n+1}} y_{n+1} + \beta, z_{n+1}) = 0, \quad (3.46)$$

$$F_2(t_{n+1}, y_{n+1}, z_{n+1}) = 0, \quad (3.47)$$

where α and β are constants and $h_{n+1} = t_{n+1} - t_n$. The corresponding Jacobian matrix is of the form

$$J = \begin{pmatrix} \frac{\alpha}{h_{n+1}} \frac{\partial F_1}{\partial y'} + \frac{\partial F_1}{\partial y} & \frac{\partial F_1}{\partial z} \\ \frac{\partial F_2}{\partial y} & \frac{\partial F_2}{\partial z} \end{pmatrix}. \quad (3.48)$$

If the index of the DAE system is not equal to 0 (i.e., it is not a pure ODE system), the Jacobian matrix is poorly conditioned when the stepsize is small because the scaled Jacobian matrix $\frac{h_{n+1}}{\alpha} J$ is close to a singular matrix of the form

$$\begin{pmatrix} \frac{\partial F_1}{\partial y'} & 0 \\ 0 & 0 \end{pmatrix}. \quad (3.49)$$

Furthermore, the initial stepsize in DASSL is chosen to be $O(TOL)$, where TOL is the user-defined tolerance. This implies that the Jacobian matrix will have a large condition number if the user supplies a sharp tolerance. A poorly conditioned Jacobian will sometimes lead to failure of the error test or the convergence test for the Newton iteration. After such an unsuccessful step, DASSL will respond by reducing the stepsize, which leads to a Jacobian matrix with an even larger condition number. This can lead to a cycle in which DASSL is unable to take a successful first step.

Petzold and Lötstedt gave a general technique for scaling the equations and variables in (3.46) that overcomes this difficulty. Since we are only interested in the index-1 case, we will only present their technique for index-1 problems. If the index of the DAE system is one, we have that $\partial F_2/\partial z$ is not singular. Therefore, if we scale the rows corresponding to the algebraic constraints by $1/h_{n+1}$, the Jacobian matrix is nonsingular as $h \rightarrow 0$. The effect of this scaling should therefore improve the accuracy of the solution.

We thus apply the above scaling technique in our Newton iteration process, solving the index-1 DAE system (3.19)-(3.21). As shown in Figure 3.3, our Newton iteration matrix consists of two decoupled ABD matrices. COLROW is applied twice to solve the two linear equation systems separately. The four subblocks, which are associated with the boundary conditions, represent the equations corresponding to the algebraic constraints. We scale these four $NPDE$ by $2NPDE$ subblocks and the corresponding rows of the right-hand side by $1/h_{n+1}$. However, for efficiency considerations, DASSL does not necessarily compute a new Jacobian matrix at each step. It is possible for DASSL to use the same Jacobian matrix (already decomposed in LU form) even when the stepsize has changed slightly. As mentioned above, we scale the subblocks before the linear system solver performs an LU decomposition of the Jacobian. Then we save this stepsize and use it to perform scaling over the corresponding components of the right-hand side. Three subroutines inside DASSL, DAINI, DASTP and DASLV, are modified to implement the proper scaling process.

3.2.8 Other Modifications to DASSL

In addition to the modifications mentioned above, (i.e., introducing COLROW as a linear system solver and scaling the Newton iteration to overcome large condition numbers,) we have made some further modifications to DASSL, which are associated with our adaptive strategy and the MOL approach.

First, within DASSL there is an option for specifying an absolute tolerance, *ATOL*, and a relative tolerance, *RTOL*. Users can choose them both as scalars or both as vectors. However, if they are set to be vectors, each of them must have the dimension $NC * NPDE$, i.e., the number of equations of the DAE system. This flexibility is unnecessary for a MOL code, which requires the same tolerance on all DAEs corresponding to the same PDE component. Therefore, we have changed the dimension of *ATOL* and *RTOL* to be *NPDE* in the case where they are to be vectors. The B-spline coefficients corresponding to the same PDE component will thus be determined to the same tolerance. This modification is limited to the routines, DASSL and DAWTS.

The normal mode of operation of DASSL is that it integrates from the start to the end without providing any intermediate output. In many situations it is useful to have information at the end of every timestep. DASSL provides an option to operate in this mode. Since the approximate spatial error is calculated after each step in BACOL, we set $INFO(3) = 1$, prior to the call to DASSL. This requires DASSL to return after each successful step. If we decide that there is no remeshing needed, we will simply recall DASSL to continue. However, if we decide to perform a remeshing and continue integration afterward, we first discard the current step and go back to the last accepted step. We do not allow DASSL to increase the stepsize for the next attempt; i.e., in the first step after a remeshing, the stepsize that is used is not allowed to be greater than the stepsize used in the last accepted time step (this is similar with the standard strategy for ODE cases after a time step failure; See, e.g., [24]). If the predicted stepsize is greater than the previous one, we set it equal to the

previous stepsize. The philosophy is that we want to be conservative in the first step after a remeshing. Based on the same reasoning, the order of the BDF method which is employed in the first step after a remeshing is not allowed to be greater than the order used in the previous successful step.

During our numerical experiments, we used the UNIX command, *gprof*, to obtain the execution time for each subroutine. (See Section 4.2 for results.) We found that DANRM, a subroutine in DASSL which computes the norm of a vector, represented about 20% of the total running time. In its double precision version, DDANRM, has only twenty lines of code, as follows:

```

      DOUBLE PRECISION FUNCTION DDANRM (NEQ, V, WT, RPAR, IPAR)
      INTEGER  NEQ, IPAR(*)
      DOUBLE PRECISION  V(NEQ), WT(NEQ), RPAR(*)
      INTEGER  I
      DOUBLE PRECISION  SUM, VMAX
C***FIRST EXECUTABLE STATEMENT  DDANRM
      DDANRM = 0.0D0
      VMAX = 0.0D0
      DO 10 I = 1,NEQ
          IF(ABS(V(I)/WT(I)) .GT. VMAX) VMAX = ABS(V(I)/WT(I))
10      CONTINUE
      IF(VMAX .LE. 0.0D0) GO TO 30
      SUM = 0.0D0
      DO 20 I = 1,NEQ
20      SUM = SUM + ((V(I)/WT(I))/VMAX)**2
      DDANRM = VMAX*SQRT(SUM/NEQ)
30      CONTINUE
      RETURN
C-----END OF FUNCTION DDANRM-----

```

END

If we count the square operator as a multiplication, there are approximately $5 * NEQ$ multiplications (a division takes roughly the same time as a multiplication), where NEQ is the length of the vector. It is interesting to see that $V(I)/WT(I)$ is calculated three times. However, if the compiler provides a sufficient level of optimization, $V(I)/WT(I)$ can be calculated only twice by storing the intermediate value in the statement

```
IF(ABS(V(I)/WT(I)) .GT. VMAX) VMAX = ABS(V(I)/WT(I)).
```

We therefore assume that there are approximately $4 * NEQ$ multiplications overall. If we can create another vector of length NEQ , which serves as a storage for $V(I)/WT(I)$, we do not need to calculate these values again. Therefore, only $3 * NEQ$ multiplications are needed; i.e., about 25% is saved in DDANRM and about 5% is saved in the total running time. We therefore used a part of $RPAR$, which is a parameter array, to save $V(I)/WT(I)$. The modification is straightforward, as follows

```
DOUBLE PRECISION FUNCTION DDANRM (NEQ, V, WT, RPAR, IPAR)
  INTEGER  NEQ, IPAR(*)
  DOUBLE PRECISION  V(NEQ), WT(NEQ), RPAR(*)
```

C

```
  INTEGER  I
  DOUBLE PRECISION  SUM, VMAX
```

C-----

```
      double precision      temp
      integer               itemp
      integer               iwkdnm
      parameter             (iwkdnm = 49)
```

c

```
      rpar(ipar(iwkdnm)) is the work storage
```

c

```
      for the modified version of the
```

```

c                               subroutine DDANRM.
c
c-----
C***FIRST EXECUTABLE STATEMENT  DDANRM
    DDANRM = 0.0D0
    VMAX = 0.0D0
    itemp = ipar(iwkdnm) - 1
    DO 10 I = 1,NEQ
c-----
        itemp = itemp + 1
        rpar(itemp) = abs(v(i)/wt(i))
        if (rpar(itemp) .gt. vmax) vmax = rpar(itemp)
c-----
10    CONTINUE
    IF(VMAX .LE. 0.0D0) GO TO 30
    SUM = 0.0D0
c-----
        itemp = ipar(iwkdnm) - 1
c-----
    DO 20 I = 1,NEQ
c-----
        itemp = itemp + 1
        temp = rpar(itemp)/vmax
        sum = sum + temp * temp
20    continue
c-----
    DDANRM = VMAX*SQRT(SUM/NEQ)
30    CONTINUE
    RETURN
```



```
C-----END OF FUNCTION DDANRM-----  
      END
```

The effect of the above modification is shown in Chapter 4. As we know, DASSL was not originally developed for MOL codes. Thus it is possible some parts of DASSL may not be optimal if the number of ODEs or DAEs is large, as is the case when the number of subintervals is large. This is an open question, which is beyond the scope of this thesis, but deserves attention in the future.

3.3 Spatial Adaptivity

In this section we first introduce an *a posteriori* spatial error estimate, and an h-refinement algorithm based on this estimate is then derived. The approximate spatial error is computed at each step, with a remeshing being done when the error estimate is larger than the tolerance or the mesh is not asymptotically equidistributed (to be described later). After the new mesh is obtained, BACOL will repeat the current time step. Instead of using a local refinement as considered by Adjerdid et al. [4], BACOL employs a global mesh refinement strategy. During the remeshing process, BACOL will also estimate the number of intervals needed for the new mesh. If the code decides that the current number of intervals is too large or too small, a remeshing is then carried out with the newly estimated number of intervals. Trouble with this process can arise when the number of intervals is very small. Section 3.3.3 will discuss an implementation which deals with this issue.

3.3.1 Spatial Error Estimation

Many successful *a posteriori* spatial error estimates for solutions of parabolic or elliptic problems are obtained by using a piecewise polynomial approximation having a higher degree than that used for the representation of the solution, e.g., [5, 16, 17]. Some

error estimation techniques have been proven to converge to the exact error [5, 57], in restricted cases, e.g., in the H^1 -norm when a fixed mesh is used.

In this thesis, the approximate solution, $U(x, t)$, is calculated in a piecewise polynomial subspace of degree p , and a second piecewise solution, $\bar{U}(x, t)$, is computed in the similar subspace of degree $p + 1$. Let variables without bars denote variables corresponding to the piecewise solution of degree p , and variables with bars denote variables corresponding to the piecewise solution of degree $p + 1$. For example, ξ_l represents the l -th collocation point for the piecewise polynomial solution of degree p , while $\bar{\xi}_l$ represents the l -th collocation point for the piecewise polynomial solution of degree $p + 1$. Thus from (3.19)-(3.21) we obtain

$$b_L(t, U(0, t), U_x(0, t)) = 0, \quad (3.50)$$

$$\sum_{m=(i-1)(p-1)+1}^{i(p-1)+2} B_m(\xi_l) y'_{m,s}(t) = f_s(t, \xi_l, U(\xi_l, t), U_x(\xi_l, t), U_{xx}(\xi_l, t)), \quad (3.51)$$

where ξ_l is one of the $p - 1$ collocation points in the i -th subinterval, $i = 1, \dots, N$, $s = 1, \dots, NPDE$,

$$b_R(t, U(1, t), U_x(1, t)) = 0, \quad (3.52)$$

$$b_L(t, \bar{U}(0, t), \bar{U}_x(0, t)) = 0, \quad (3.53)$$

$$\sum_{m=(i-1)p+1}^{i p+2} \bar{B}_m(\bar{\xi}_l) \bar{y}'_{m,s}(t) = f_s(t, \bar{\xi}_l, \bar{U}(\bar{\xi}_l, t), \bar{U}_x(\bar{\xi}_l, t), \bar{U}_{xx}(\bar{\xi}_l, t)), \quad (3.54)$$

where $\bar{\xi}_l$ is one of the p collocation points in the i -th subinterval, $i = 1, \dots, N$, $s = 1, \dots, NPDE$,

$$b_R(t, \bar{U}(1, t), \bar{U}_x(1, t)) = 0. \quad (3.55)$$

A natural way to treat (3.50)-(3.55) is as two separate DAE systems. However, this would require running two copies of DASSL, and it would mean that different time steps could be employed for the computation of the two solutions. Thus interpolation

of one of the solutions would be unavoidable, during the calculation of the spatial error estimate at a given time. This would considerably increase the complexity of the algorithm. More seriously, this would make it difficult to continue the integration after a remeshing, because the values of the two solutions at the previous steps would have to be interpolated from two different sources. After a remeshing, the error estimate (to be discussed shortly) based on the two solutions on the new mesh might be large enough to cause a failure of the spatial error test. For these reasons, we will treat (3.50)-(3.55) as a single DAE system. Then no interpolation in time is needed and, after a remeshing, both solutions at the previous steps on the new mesh can be interpolated using the piecewise solution of degree $p + 1$. Equation (3.45) becomes

$$\begin{aligned} U^*(\xi_l^*, t_{n-i}) &= \bar{U}(\xi_l^*, t_{n-i}), & l = 1, \dots, NC^*, \\ \bar{U}^*(\bar{\xi}_l^*, t_{n-i}) &= \bar{U}(\bar{\xi}_l^*, t_{n-i}), & l = 1, \dots, \bar{N}C^*, \end{aligned}$$

where $i = 0, \dots, k$ and k is the order of the BDF method.

An *a posteriori* spatial error estimate is obtained by comparing the two solutions appearing in (3.50)-(3.55). Let $ATOL_s$ and $RTOL_s$ denote the absolute and relative tolerance for the s -th component of the PDE system. Using a norm defined as follows, the normalized error estimate for the s -th PDE component over the whole problem interval $[0, 1]$, is given by

$$\|E_s\| = \sqrt{\int_0^1 \left(\frac{U_s - \bar{U}_s}{ATOL_s + RTOL_s |U_s|} \right)^2 dx}, \quad (3.56)$$

while the normalized error estimate, \hat{E}_i , for the i -th subinterval, is calculated from

$$\|\hat{E}_i\| = \sqrt{\sum_{s=1}^{NPDE} \int_{x_{i-1}}^{x_i} \left(\frac{U_s - \bar{U}_s}{ATOL_s + RTOL_s |U_s|} \right)^2 dx}, \quad (3.57)$$

where $i = 1, \dots, N$.

Spatial Error Test I

The approximate error is computed at each successful time step. As well we compute the parameters

$$r_1 = \max_{1 \leq i \leq N} \|\hat{E}_i\|^{1/(p+1)} \quad \text{and} \quad r_2 = \frac{\sum_{i=1}^N \|\hat{E}_i\|^{1/(p+1)}}{N}. \quad (3.58)$$

We note that r_1 represents a measure of the maximum subinterval error estimate while r_2 represents the average. The ratio r_1/r_2 gives an indicator of the error distribution over the mesh subintervals. Specifically, if it is large, the maximum subinterval error estimate is significantly larger than the average one and the mesh is not well-distributed. At each step, BACOL checks to see if

$$\frac{r_1}{r_2} \leq 2. \quad (3.59)$$

We call a mesh which satisfies (3.59) an *asymptotically equidistributed* mesh (which is slightly different from the definition in Page 364, [7]).

If it is the initial time step, or if (3.59) is not satisfied, or if

$$\|E\| = \max_{0 \leq s \leq NPDE} \|E_s\| \geq 1, \quad (3.60)$$

an adaptive strategy (to be described shortly) is invoked. We note that if $RTOL_s = 0$, $\|E_s\| > 1$ implies that the L^2 -norm error estimate is larger than the absolute tolerance, $ATOL_s$, in the s -th component of PDEs. We call (3.59), (3.60), *spatial error test I*.

3.3.2 The Remeshing Strategy

Computational evidence [70] indicates that, by using a collocation method based on B-splines (EPDCOL [70]) with a fixed uniform mesh, the L^2 -norm error is $O(h^{p+1})$, where $h = \max_{1 \leq i \leq N} h_i$. Let $U_s(x, t)$ and $\bar{U}_s(x, t)$ be the two approximate solutions for the s -th PDE component, as discussed in the previous subsection. Numerical

results in [70] show that the L^2 -norm error estimate for $[x_{i-1}, x_i]$,

$$\|U_s(x, t) - \bar{U}_s(x, t)\|_2 = \sqrt{\int_{x_{i-1}}^{x_i} (U_s(x, t) - \bar{U}_s(x, t))^2 dx}$$

converges to the exact error in the L^2 -norm as $h \rightarrow 0$,

$$\|U_s(x, t) - u_s(x, t)\|_2 = \sqrt{\int_{x_{i-1}}^{x_i} (U_s(x, t) - u_s(x, t))^2 dx},$$

where $s = 1, \dots, NPDE$ and u_s represents the exact solution for the s -th component of PDEs.

Numerical experiments in Section 5.6 indicate the order of convergence in L^2 -norm is equal to $p + 1$, i.e.,

$$\|E\|_2 = O(h^{p+1}) \leq Ch^{p+1},$$

where $h_i = x_i - x_{i-1}$, $h = \max_{1 \leq i \leq N} h_i$ and $\|\cdot\|_2$ is the L^2 -norm. We note that

$$\begin{aligned} \|E\| &\leq \frac{1}{\min_{1 \leq s \leq NPDE} ATOL_s} \|E\|_2 \\ &\leq \frac{C}{\min_{1 \leq s \leq NPDE} ATOL_s} h^{p+1}. \end{aligned}$$

We develop our remeshing strategy based on the approximate error formulas (3.56) (3.57) and the assumption that

$$\|\hat{E}_i\| \propto h_i^{p+1}, \quad (3.61)$$

$$\|E\| \propto h^{p+1}, \quad (3.62)$$

where $h_i = x_i - x_{i-1}$, $h = \max_{1 \leq i \leq N} h_i$.

For boundary value problems for ODEs, we have the following result (e.g., [7]). When $N \rightarrow \infty$, using a quasiuniform mesh (i.e., all meshes satisfying $h \leq Kh_{\min}$, where $h_{\min} := \min_{1 \leq i \leq N} h_i$, for some fixed constant K), a numerical method of order $p + 1$ yields a $O(N^{-(p+1)})$ error, i.e.,

$$\max_{1 \leq s \leq NPDE} |u_s(x, t) - U_s(x, t)| = O(N^{-(p+1)}),$$

where $u_s(x, t)$ is the exact solution of the s -th PDE component. We will now assume that the following property,

$$\|E\| \propto N^{-(p+1)}, \quad (3.63)$$

holds for PDEs.

As indicated earlier, at the initial time step or any step at which either (3.59) or (3.60) is satisfied, our remeshing strategy is invoked.

The remeshing strategy is a critical component of our algorithm. Ascher et al. describe the following paragraph on page 380 of [7].

One of the most important things to bear in mind when one is producing a nontrivial implementation of mesh selection is this: Before adding sophisticated precautions, it is important that one first have a good understanding of the key elements of the basic strategy.

We will now discuss a number of details associated with our remeshing algorithm.

Spatial Error Test II

We first note a fact that if the number of intervals is not large enough, frequent remeshing can happen, i.e., there will be only a few time steps between remeshings. This may cause serious difficulty in that DASSL may be unable to continue the time integration. For instance, if there are only 2 successful steps between 2 remeshings and DASSL is using a 5-th order BDF method, then in order to continue the integration in an efficient manner, BACOL will need to perform interpolation of the solution values from up to 6 previous time steps; i.e., y_{n-i} , $i = 0, \dots, 5$. Consequently, the residual may not be sufficiently accurate because y_{n-i} , $i = 2, \dots, 5$, will have been interpolated twice; i.e., they have been interpolated in both remeshings. Then DASSL may be unable to continue integration with the same order and the same stepsize.

More seriously, DASSL may fail completely; i.e., it may not be able to restart even at order 1 and with a very small stepsize.

A second difficulty arises when the number of intervals is greater than necessary, since BACOL will then take a considerable number of time steps before the next remeshing. This is undesirable because it is inefficient to solve an unnecessarily large ODE or DAE system. It is difficult to tell how long we should continue the time integration with the current mesh before performing the next remeshing. Fortunately, we have noted that between two consecutive remeshings, the normalized error estimate, in most cases, increases gradually. Therefore, if we can arrange that the normalized error estimate for the first step after a remeshing is in a suitable range, $[c_1, c_2]$, $0 < c_1 < c_2 < 1$, the next remeshing will be performed after a reasonable number of time steps. We will thus consider the corresponding number of mesh points to be *approximately optimal*. In BACOL, we let $c_1 = 0.1$ and $c_2 = 0.4$. We use a mesh, $\{x_i\}_{i=1}^{N^*}$, where N^* is the predicted number of mesh points (the strategy to predict the number of intervals will be discussed shortly). After a successful step by DASSL, if the normalized error estimate, $\|E\|$, satisfies *spatial error test II*, defined as follows,

$$\frac{r_1}{r_2} \leq 2 \quad \text{and} \quad (3.64)$$

$$0.1 < \|E\| < 0.4, \quad (3.65)$$

then we accept the current step. Otherwise, we reject the step.

Predicting N^*

We assume that BACOL (using DASSL) has taken a time step and that the normalized spatial error estimate has been computed. We also assume that the spatial error test *I* has indicated that a remeshing is required. Our action depends on how many times we have attempted a remeshing for the current time.

- If it is the first remeshing at the current step, we use the same number of intervals; i.e., redistribute the mesh without changing the number of intervals.

The philosophy is that most of time when the spatial error test I fails it is not because more intervals are needed, but is because the current mesh is not well distributed.

- If it is not the first or fifth attempted remeshing, BACOL will predict the number of intervals suitable for the current time, using a strategy to be discussed shortly.
- If it is the fifth attempted remeshing, then BACOL will perform a cold start with the same number of intervals as the last accepted step. This idea comes from the belief that if BACOL has failed to pass the spatial error test II after four attempted remeshings, then the error estimates from the previous time, where a BDF method with order greater than 1 was used, are likely not reliable.

Suppose that we perform a remeshing with a predicted N^* value but our approximate solution still fails the spatial error test II . Our experience shows that most of the time the mesh is well distributed so that (3.64) is satisfied but the normalized error estimate fails to satisfy (3.65). From (3.63), we have

$$\|E\| \propto N^{-(p+1)}.$$

We next require that the normalized error estimate after remeshing, $\|E^*\|$, will be equal to 0.2. Then we have

$$\|E\| \propto N^{-(p+1)}, \quad (3.66)$$

$$0.2 = \|E^*\| \propto (N^*)^{-(p+1)}. \quad (3.67)$$

Dividing (3.66) by (3.67), we obtain

$$\frac{\|E\|}{0.2} \propto \left(\frac{N^*}{N}\right)^{p+1}. \quad (3.68)$$

In our remeshing strategy, we set

$$\frac{\|E\|}{0.2} = \left(\frac{N^*}{N}\right)^{p+1}. \quad (3.69)$$

This gives us N^* , the predicted number of intervals for the current time as follows.

$$N^* = N \left(\frac{\|E\|}{0.2} \right)^{1/(p+1)}. \quad (3.70)$$

We note that if $\|E\| > 0.4$, N^* will be greater than N ; on the other hand, if $\|E\| < 0.1$, N^* will be less than N .

Global Mesh Refinement

Once the new number of intervals N^* is determined based on the equidistribution principle, BACOL tries to generate a new mesh, $\{x_i^*\}_{i=1}^{N^*}$, which satisfies

$$\left(\sqrt{\sum_{s=1}^{NPDE} \int_{x_{i-1}^*}^{x_i^*} \left(\frac{U_s - \bar{U}_s}{ATOL_s + RTOL_s |U_s|} \right)^2 dx} \right)^{\frac{1}{p+1}} = \text{constant}, \quad (3.71)$$

where the constant in (3.71) is

$$\frac{\sum_{i=1}^N \|\hat{E}_i\|^{1/(p+1)}}{N^*}.$$

This process is straightforward since we assume that $\|\hat{E}_i\|$ is uniform $[x_{i-1}, x_i]$.

Algorithm Summary

(The first step is a special case: we apply the spatial error test II , and if it fails, we predict N^* using (3.70) and apply a cold start.) We now present the algorithm summary of BACOL as follows.

- 1) Assume that we have taken a step from t_i to t_{i+1} . If this is the first or fifth attempted remeshing for t_i , set $N^* = N$; otherwise predict N^* from (3.70).
- 2) Determine a new mesh, with N^* determined from 1), using the global mesh redistribution algorithm based on (3.71).
- 3) Interpolate solution values at current and possibly previous time steps from the previous mesh to the new mesh.

- 4) If this is one of the first four attempted remeshings, perform a continuation using DASSL from t_i ; otherwise, perform a cold start using DASSL from t_i .
- 5) Compute the normalized spatial error estimate and apply spatial error test *II*.
- 6) If spatial error test *II* is satisfied, accept this step and continue with DASSL; otherwise, reject this step and go back to 1).

3.3.3 Cycle Avoidance

In this subsection we consider a potential danger of our remeshing strategy, and a modification of our implementation to avoid this difficulty.

If the number of intervals necessary is very small and the order of the collocation method is high, our remeshing strategy may have trouble finding a suitable number of intervals. For instance, assume $N = 4$, $p = 9$, and $\|E\| = 0.5$. The new number of intervals is then computed by (3.71). We then obtain $N^* \approx 4.38$. If such a situation arises (the code indicates that more points are necessary but $0 < N^* - N < 1$), BACOL will let $N^* = N + 1$. However, if we try $N^* = 5$, a simple calculation, using the formula

$$\frac{\|E\|}{\|E^*\|} = \left(\frac{N^*}{N}\right)^{p+1}, \quad (3.72)$$

shows that $\|E^*\| \approx 0.067$. Therefore, BACOL will remesh again because $\|E^*\| < 0.1$. The required number of intervals will be predicted again; this calculation will give 4.38 again. If the code indicates that fewer intervals are needed but $0 < N - N^* < 1$, BACOL will set $N^* = N - 1$. Thus N^* will be 4 again. We call this undesirable behavior a *cycle*. Fortunately, we have observed that this only happens when N is very small and p is large, and when the solution is easy to compute. In this situation, the execution time is typically small, and we can afford to use a slightly larger number of subintervals than predicted. Assuming that we have $N^* = N + 1$, we need to determine what range of N values leads to a situation where

$$\|E\| > 0.4 \quad \text{and} \quad \|E^*\| < 0.1.$$

To answer this question, we once again use (3.72) and obtain

$$4 < \frac{\|E\|}{\|E^*\|} = \left(\frac{N+1}{N}\right)^{p+1}. \quad (3.73)$$

We recall that BACOL requires the degree of polynomial, p , to be in the range $3 \leq p \leq 11$, i.e., the maximum value of p is 11. Substituting $p = 11$ into (3.73), we see that a cycle could happen if

$$N < 8.17. \quad (3.74)$$

Therefore in BACOL we set a parameter $\hat{N} = 15$ and impose the condition that if $N \leq \hat{N}$, the spatial error test II will be modified to be

$$\frac{r_1}{r_2} \leq 2 \quad \text{and} \quad (3.75)$$

$$\|E\| < 0.4, \quad (3.76)$$

i.e. if N is less than or equal to 15, we do not care too much about whether N is possibly larger than necessary since the associated computation will not be expensive.

3.3.4 The Remeshing Algorithm

We now present the overall remeshing strategy as follows:

- 10 if (the computation has reached the output time) go to 30
- 20 take next time step
- if ($r_1/r_2 > 2$) then
 - reject the current step and perform a remeshing
 - go to 20
- else
 - if (it is the initial step) or (it is the first step after remeshing) then
 - if ($0.1 < \|E\| < 0.4$) then
 - accept the current step and continue the integration
 - go to 10

```
    else
        reject the current step and perform a remeshing
        go to 20
    endif
else
    if ( $\|E\| < 1$ ) then
        accept the current step and continue the integration
        go to 10
    else
        reject the current step and perform a remeshing
        go to 20
    endif
endif
endif
30 stop
```

Chapter 4

Description of the BACOL Software

This chapter first describes the subroutines included in the BACOL software package. Then in Section 4.2 the user supplied subroutines are discussed. A sample program is given in Section 4.3. Finally in Section 4.4 the structure of the BACOL software is described and a profile of computer time usage for the main subroutines is presented.

4.1 A Description of all Subroutines

This section describes the four main components of the BACOL software package, including the driver subroutine, the core integrator, miscellaneous subroutines for setting up and solving the nonlinear equations, and the subroutines for remeshing. As indicated earlier, a number of the important computational tasks performed in BACOL are implemented using existing packages; we will consider these packages later in this section. We first introduce the subroutines which are new.

BACOL. This routine is the driver for the entire package. It first checks the user's input for legality and performs initialization tasks such as setting parameters

and allocating and defining the locations and lengths of the storage array required by the package. It also calls COLPNT, INIY, INIYP, and MESHSQ to complete the initialization. After that, BACOL makes repeated calls to the core integrator DASSL in order to take time steps. After each step, the spatial error estimate is computed by calling ERREST. If the spatial estimate is satisfactory then this step is accepted and SUCSTP is called to update the current information in case there is a remeshing in the future. On the other hand, if the spatial error estimate is unsatisfactory, this step is rejected and BACOL calls REMESH, REINIT, and DIVDIF to carry out a remeshing; this may involve modification of the location and lengths of the storage array. After a remeshing, BACOL will call DASSL again, beginning at the last successful step. This process is repeated until a time step is accepted or an error condition occurs. When the output time T_{out} is reached, BACOL returns the B-spline coefficients which are used by the VALUES routine to calculate the values of $U(x, T_{out})$ for any x , $x_a \leq x \leq x_b$.

VALUES. This is the subroutine that the user must call in order to obtain the values of the approximate solution and derivatives at T_{out} and at any given spatial points. When the BACOL software performs a remeshing, VALUES is also called within BACOL to obtain the values at the new collocation points at the current step and previous steps which are required for a continuation of the integration. This subroutine calls the B-spline subroutines BSPLVD and INTERV.

COLPNT. This subroutine calculates the Gauss-Legendre collocation point sequence, $\{\xi_i\}_{i=1}^{NC}$.

INIY, INIYP, MESHSQ. The main purposes of these subroutines are to determine consistent initial conditions for DASSL, to generate the values of the piecewise polynomial and its first and second spatial derivatives at the collocation points, and to prepare for calculating a future spatial error estimate.

MESHSEQ calculates the spatial mesh size sequence and generates the points and weights for the Gaussian quadrature rule used in the spatial error estimate. INIY calculates the N subblocks for $\frac{\partial F}{\partial y}$ shown in Figure 3.1 and determines the values of the B-spline coefficients at $t = 0$, $y(0)$. INIYP computes the first derivatives, $y'(0)$. Combining $y(0)$ with $y'(0)$ will give consistent initial conditions for DASSL.

ERREST. This subroutine computes the spatial error estimate for each subinterval, and for each component of the PDEs. It also decides whether or not a remeshing is necessary.

ERRVAL. This subroutine computes the approximate solutions at the Gaussian points when a Gaussian quadrature rule is applied to obtain the values of the integral in the formulas for the spatial error estimates, (3.56) and (3.57). It is called by ERREST.

SUCSTP. After each accepted time step, this subroutine stores the information which is necessary if a remeshing is required at a later step.

REMESH, REINIT, DIVDIF. These subroutines implement the remeshing process, a very important component of the BACOL package. REMESH generates a new mesh by equidistributing the error measure, using (3.71). REINIT prepares for the continuation of the time integration after a remeshing; tasks include: calculating the spatial mesh step size sequence and the collocation point sequence, computing the values of the piecewise polynomial functions and their first and second derivatives at the collocation points, and computing the B-spline coefficients at the previous steps. DIVDIF generates the divided differences which are required by DASSL.

JAC, CALJAC. These two subroutines compute the Jacobian matrices. We recall that two ABD matrices are combined in a decoupled form to represent the

Jacobian matrix. Therefore, JAC calls CALJAC twice and each time CALJAC computes one of the ABD matrices.

RES, CALRES. These two subroutines compute the residual of a Newton system as required by DASSL. As above, RES calls CALRES twice and each time CALRES computes the residual corresponding to one of the ABD matrices.

EVAL. The purpose of this routine is to evaluate the piecewise polynomial functions and their first and second derivatives at a given collocation point. The difference between VALUES and EVAL is that VALUES is able to compute the piecewise polynomial solution at *any* point or points and for *any* derivatives which are less than or equal to p . Furthermore, VALUES can also estimate these values at previous time steps while EVAL can only calculate the values for the current step.

We now briefly describe the subroutines which are included in BACOL but were developed by others.

DASSL The DASSL package, developed by Petzold [61], is the core time integrator for the BACOL package. As discussed in Chapter 3, some modifications have been made to DASSL.

BSPLVD, BSPLVN, INTERV. These subroutines, part of the B-spline package developed by De Boor [21, 22], are used to generate the values of the B-spline basis functions and their derivatives at any desired points.

CRDCMP, CRSLVE. These subroutines comprise the COLROW package written by Diaz et al. [30]. CRDCMP is used to factor the ABD matrix and CRSLVE is used to solve the corresponding linear system.

GAULEG. This subroutine was developed by Keast [45]. It calculates the points and weights for a Gauss-Legendre or Gauss-Lobatto quadrature rule over the interval $[-1, 1]$ or $[0, 1]$. It is called by COLPNT and MESHQS.

4.2 User Supplied Subroutines

The user is required to provide seven subroutines which define the form of the PDE problem. For given input values of x and t and corresponding input values of U , U_x , and U_{xx} , the subroutines do the following tasks:

F. This subroutine computes the values, $f_s(t, x, U(x, t), U_x(x, t), U_{xx}(x, t))$, $s = 1, \dots, NPDE$, representing the right-hand side of (3.10). It is needed in the calculation of the Newton residuals, and for setting up the collocation equations.

DERIVF. This subroutine is used to provide the information required in (3.40) to form the analytic Jacobian matrix for the DAE system. It computes $\partial f/\partial U$, $\partial f/\partial U_x$ and $\partial f/\partial U_{xx}$.

BNDXA. This subroutine computes the boundary condition at the left boundary point x_a , namely, $b_L(t, U(x_a, t), U_x(x_a, t))$, which is needed in the calculation of the Newton residuals.

BNDXB. This subroutine computes the boundary condition at the right boundary point x_b , namely, $b_R(t, U(x_b, t), U_x(x_b, t))$, which is needed in the calculation of the Newton residuals.

DIFBXA. This subroutine is used to define the differentiated boundary condition at the left boundary point x_a . It computes $\partial b_L/\partial U$ and $\partial b_L/\partial U_x$ which are needed in (3.36)-(3.39) for the analytic Jacobian matrix, and $\partial b_L/\partial t$ which is needed in (3.41)-(3.44) to calculate consistent initial conditions, $y'(0)$.

DIFBXB. This subroutine is used to define the differentiated boundary condition at the right boundary point x_b . It computes $\partial b_R/\partial U$ and $\partial b_R/\partial U_x$ which are needed in (3.36)-(3.39) for the analytic Jacobian matrix, and $\partial b_R/\partial t$ which is needed in (3.41)-(3.44) to calculate consistent initial conditions, $y'(0)$.

UNIT. This subroutine computes the initial values $U(x, t_0)$ for any given x .

The user-supplied subroutines are usually easily constructed and an example showing these routines is presented in the following section.

4.3 Sample Program

To illustrate the use of BACOL, we present a driver program and the user-supplied subroutines to solve Burgers' equation

$$\begin{aligned} u_t &= -uu_x + \epsilon u_{xx}, & 0 < x < 1, & \quad t > 0, \\ u(x, 0) &= 0.5 - 0.5 \tanh\left(\frac{1}{4\epsilon}(x - 0.25)\right), & 0 \leq x \leq 1, \\ u(0, t) &= 0.5 - 0.5 \tanh\left(\frac{1}{4\epsilon}(-t - 0.25)\right), & t \geq 0, \\ u(1, t) &= 0.5 - 0.5 \tanh\left(\frac{1}{4\epsilon}(0.75 - t)\right), & t \geq 0, \end{aligned}$$

where $\epsilon = 10^{-3}$. The input parameters are chosen as follows: both the absolute tolerance, *ATOL*, and the relative tolerance, *RTOL*, are chosen to be scalars, and we set $ATOL = RTOL = 10^{-4}$; the degree of the piecewise polynomial is chosen as 3; i.e., there are 2 collocation points in each subinterval; the initial mesh is chosen as the uniform mesh where the number of subintervals is 10; the output time T_{out} is equal to 1. The approximate solution values at T_{out} are calculated at x values corresponding to 101 uniformly distributed points in $[0, 1]$ and are compared with the exact solution.

We first present the main program.

```

C-----
C CONSTANTS:
      INTEGER                KCOL
C      KCOL IS THE NUMBER OF COLLOCATION POINTS
C      TO BE USED IN EACH SUBINTERVAL, WHICH IS
C      EQUAL TO THE DEGREE OF THE PIECEWISE
C      POLYNOMIALS MINUS ONE.
```

```

C          1 < KCOL < 11.
PARAMETER (KCOL = 2)
INTEGER   NPDE
C          NUMBER OF PDES
PARAMETER (NPDE = 1)
INTEGER   NINTMX
C          MAXIMAL NUMBER OF INTERVALS ALLOWED
PARAMETER (NINTMX = 2000)
INTEGER   MAXVEC
C          THE DIMENSION OF THE VECTOR OF
C          BSPLINE COEFFICIENTS
PARAMETER (MAXVEC = NPDE*(NINTMX*KCOL+2))
INTEGER   LRP
C          SEE THE COMMENT FOR RPAR
PARAMETER (LRP = 134+NINTMX*(35+35*KCOL+31*NPDE
+          +38*NPDE*KCOL+8*KCOL*KCOL)+14*KCOL
+          +79*NPDE+NPDE*NPDE*(21
+          +4*NINTMX*KCOL*KCOL+12*NINTMX*KCOL
+          +6*NINTMX))
C          INTEGER   LIP
C          SEE THE COMMENT FOR IPAR
PARAMETER (LIP = 115+NPDE*((2*KCOL+1)*NINTMX+4))
INTEGER   LENWRK
C          THE DIMENSION OF WORK ARRAY WHEN WE
C          CALL VALUES
PARAMETER (LENWRK = (KCOL+2)+KCOL*(NINTMX+1)+4)
INTEGER   NUOUT
C          THE DIMENSION OF UOUT

```

	PARAMETER	(NUOUT = NPDE*101)
	DOUBLE PRECISION	XA
C		THE LEFT BOUNDARY POINT
	PARAMETER	(XA = 0.0D0)
	DOUBLE PRECISION	XB
C		THE RIGHT BOUNDARY POINT
	PARAMETER	(XB = 1.0D0)
C	-----	
	DOUBLE PRECISION	TO
C		TO < TOUT IS THE INITIAL TIME.
C		
	DOUBLE PRECISION	TOUT
C		TOUT IS THE DESIRED FINAL OUTPUT TIME.
C		
	DOUBLE PRECISION	ATOL(NPDE)
C		ATOL IS THE ABSOLUTE ERROR TOLERANCE
C		REQUEST AND IS A SCALAR QUANTITY IF
C		MFLAG(2) = 0.
C		
	DOUBLE PRECISION	RTOL(NPDE)
C		RTOL IS THE RELATIVE ERROR TOLERANCE
C		REQUEST AND IS A SCALAR QUANTITY IF
C		MFLAG(2) = 0.
C		
	INTEGER	NINT
C		NINT IS THE NUMBER OF SUBINTERVALS
C		DEFINED BY THE SPATIAL MESH X.
C		

```
DOUBLE PRECISION      X(NINTMX+1)
C
C      X IS THE SPATIAL MESH WHICH DIVIDES THE
C      INTERVAL [X_A,X_B] AS: X_A = X(1) <
C      X(2) < X(3) < ... < X(NINT+1) = X_B.
C

INTEGER              MFLAG(5)
C
C      THIS VECTOR OF USER INPUT DETERMINES
C      THE INTERACTION OF BACOL WITH DASSL.
C

WORK STORAGE:
DOUBLE PRECISION      RPAR(LRP)
C
C      RPAR IS A FLOATING POINT WORK ARRAY
C      OF SIZE LRP.
C

INTEGER              IPAR(LIP)
C
C      IPAR IS AN INTEGER WORK ARRAY
C      OF SIZE LIP.
C
-----
DOUBLE PRECISION      Y(MAXVEC)
C
C      ON SUCCESSFUL RETURN FROM BACOL, Y IS
C      THE VECTOR OF BSPLINE
C      COEFFICIENTS AT THE CURRENT TIME TO.
C

INTEGER              IDID
C
C      IDID IS THE BACOL EXIT STATUS FLAG
C      WHICH IS BASED ON THE EXIT STATUS FROM
C      DASSL ON ERROR CHECKING PERFORMED BY
C      BACOL ON INITIALIZATION.
```

```

C-----
      DOUBLE PRECISION      EXACTU(NPDE)
C                               EXACT SOLUTION AT A CERTAIN POINT USED
C                               ONLY IN EXPERIMENTS WHERE IT IS KNOWN.
      DOUBLE PRECISION      UOUT(NUOUT)
C                               THE APPROXIMATION SOLUTIONS AT A SET
C                               OF POINTS.
      DOUBLE PRECISION      VALWRK(LENWRK)
C                               VALWRK IS A WORK ARRAY IN VALUES.
      DOUBLE PRECISION      XOUT(101)
C                               XOUT IS A SET OF SPATIAL POINTS FOR
C                               OUTPUT.
      DOUBLE PRECISION      COEFF
C                               COEFF IS THE COEFFICIENT OF UXX IN THE
C                               BURGERS' EQUATION.
      COMMON /BURGER/      COEFF
      INTEGER                I

```

```

C-----
C SUBROUTINES CALLED:
C                               BACOL
C                               VALUES
C                               TRUU
C-----

```

```

C   SET THE REMAINING INPUT PARAMETERS.
      TO = 0.0D0
      TOUT = 1.0D0
      ATOL(1) = 1.D-4
      RTOL(1) = ATOL(1)

```

```
NINT = 10
COEFF = 1.D-3

C   DEFINE THE MESH BASED ON A UNIFORM STEP SIZE.
X(1) = XA
DO 10 I = 2, NINT
    X(I) = XA + ((I-1) * (XB - XA)) / NINT
10 CONTINUE
X(NINT+1) = XB

C   INITIALIZE THE MFLAG VECTOR.
DO 20 I = 1, 4
    MFLAG(I) = 0
20 CONTINUE
MFLAG(5) = 1

WRITE(6, '( /A, I3, A, I4, 2(A, E8.2))') 'KCOL =', KCOL, ' NINT =',
& NINT, ' ATOL(1) =', ATOL(1), ' RTOL(1) =', RTOL(1)
WRITE(6, '( /A, E8.2)') 'EPS =', COEFF

CALL BACOL(TO, TOUT, ATOL, RTOL, NPDE, KCOL, NINTMX, NINT, X,
& MFLAG, RPAR, LRP, IPAR, LIP, Y, IDID)

C   CHECK FOR AN ERROR FROM BACOL.
WRITE(6, '(A, I5)') 'IDID =', IDID
IF (IDID .LT. 2) GOTO 100

XOUT(1) = XA
DO 30 I = 2, 100
```

```

      XOUT(I) = XA + DBLE(I - 1) * (XB - XA)/100.DO
30  CONTINUE
      XOUT(101) = XB

      CALL VALUES(KCOL, XOUT, NINT, X, NPDE, 101, 0, 1, UOUT, Y, VALWRK)

      DO 40 I = 1, 101
          CALL TRUU(TOUT, XOUT(I), EXACTU, NPDE)
          WRITE(6, '(/3E18.6)') XOUT(I), UOUT(I), EXACTU(1)
40  CONTINUE

      GOTO 9999

100 CONTINUE
      WRITE(6, '(A)') 'CANNOT PROCEED DUE TO ERROR FROM BACOL.'

9999 STOP
      END

```

We next present the user-supplied subroutines. The user interface for BACOL is similar to the one of EPDCOL and MSCPDE [60]. As a result, the subroutines describing the problem are the same as those in MSCPDE, and they are taken from [60].

```

C-----
      SUBROUTINE F(T, X, U, UX, UXX, FVAL, NPDE)
C-----
C PURPOSE:
C   THIS SUBROUTINE DEFINES THE RIGHT HAND SIDE VECTOR OF THE
C   NPDE DIMENSIONAL PARABOLIC PARTIAL DIFFERENTIAL EQUATION
C           UT = F(T, X, U, UX, UXX).

```



```
C
C-----
C SUBROUTINE PARAMETERS:
C INPUT:
      INTEGER              NPDE
C                          THE NUMBER OF PDES IN THE SYSTEM.
C
      DOUBLE PRECISION    T
C                          THE CURRENT TIME COORDINATE.
C
      DOUBLE PRECISION    X
C                          THE CURRENT SPATIAL COORDINATE.
C
      DOUBLE PRECISION    U(NPDE)
C                          U(1:NPDE) IS THE APPROXIMATION OF THE
C                          SOLUTION AT THE POINT (T,X).
C
      DOUBLE PRECISION    UX(NPDE)
C                          UX(1:NPDE) IS THE APPROXIMATION OF THE
C                          SPATIAL DERIVATIVE OF THE SOLUTION AT
C                          THE POINT (T,X).
C
      DOUBLE PRECISION    UXX(NPDE)
C                          UXX(1:NPDE) IS THE APPROXIMATION OF THE
C                          SECOND SPATIAL DERIVATIVE OF THE
C                          SOLUTION AT THE POINT (T,X).
C
C OUTPUT:
      DOUBLE PRECISION    FVAL(NPDE)
```

C FVAL(1:NPDE) IS THE RIGHT HAND SIDE
 C VECTOR F(T, X, U, UX, UXX) OF THE PDE.

C-----
 C DOUBLE PRECISION COEFF
 C COMMON /BURGER/ COEFF

C-----
 C
 C ASSIGN FVAL(1:NPDE) ACCORDING TO THE RIGHT HAND SIDE OF THE PDE
 C IN TERMS OF U(1:NPDE), UX(1:NPDE), UXX(1:NPDE).

C
 C FVAL(1) = COEFF*UXX(1) - U(1)*UX(1)

C
 C RETURN
 C END

C-----
 C SUBROUTINE DERIVF(T, X, U, UX, UXX, DFDU, DFDUX, DFDUXX, NPDE)

C-----
 C PURPOSE:
 C THIS SUBROUTINE IS USED TO DEFINE THE INFORMATION ABOUT THE
 C PDE REQUIRED TO FORM THE ANALYTIC JACOBIAN MATRIX FOR THE DAE
 C OR ODE SYSTEM. ASSUMING THE PDE IS OF THE FORM
 C
$$U_T = F(T, X, U, UX, UXX)$$

 C THIS ROUTINE RETURNS THE JACOBIANS $D(F)/D(U)$, $D(F)/D(UX)$, AND
 C $D(F)/D(UXX)$.

C-----
 C SUBROUTINE PARAMETERS:

C INPUT:
 C INTEGER NPDE

```
C          THE NUMBER OF PDES IN THE SYSTEM.
C
C          DOUBLE PRECISION          T
C          THE CURRENT TIME COORDINATE.
C
C          DOUBLE PRECISION          X
C          THE CURRENT SPATIAL COORDINATE.
C
C          DOUBLE PRECISION          U(NPDE)
C          U(1:NPDE) IS THE APPROXIMATION OF THE
C          SOLUTION AT THE POINT (T,X) .
C
C          DOUBLE PRECISION          UX(NPDE)
C          UX(1:NPDE) IS THE APPROXIMATION OF THE
C          FIRST SPATIAL DERIVATIVE OF THE
C          SOLUTION AT THE POINT (T,X) .
C
C          DOUBLE PRECISION          UXX(NPDE)
C          UXX(1:NPDE) IS THE APPROXIMATION OF THE
C          SECOND SPATIAL DERIVATIVE OF THE
C          SOLUTION AT THE POINT (T,X) .
C
C OUTPUT:
C          DOUBLE PRECISION          DFDU(NPDE, NPDE)
C          DFDU(I,J) IS THE PARTIAL DERIVATIVE
C          OF THE I-TH COMPONENT OF THE VECTOR F
C          WITH RESPECT TO THE J-TH COMPONENT
C          OF THE UNKNOWN FUNCTION U.
C
```

```

      DOUBLE PRECISION      DFDUX(NPDE, NPDE)
C      DFDUX(I, J) IS THE PARTIAL DERIVATIVE
C      OF THE I-TH COMPONENT OF THE VECTOR F
C      WITH RESPECT TO THE J-TH COMPONENT
C      OF THE FIRST SPATIAL DERIVATIVE OF THE
C      UNKNOWN FUNCTION U.
C
      DOUBLE PRECISION      DFDUXX(NPDE, NPDE)
C      DFDUXX(I, J) IS THE PARTIAL DERIVATIVE
C      OF THE I-TH COMPONENT OF THE VECTOR F
C      WITH RESPECT TO THE J-TH COMPONENT
C      OF THE SECOND SPATIAL DERIVATIVE OF THE
C      UNKNOWN FUNCTION U.
      DOUBLE PRECISION COEFF
      COMMON /BURGER/ COEFF
C-----
C
C      ASSIGN DFDU(1:NPDE, 1:NPDE), DFDUX(1:NPDE, 1:NPDE), AND
C      DFDUXX(1:NPDE, 1:NPDE) ACCORDING TO THE RIGHT HAND SIDE OF THE PDE
C      IN TERMS OF U(1:NPDE), UX(1:NPDE), UXX(1:NPDE).
C
      DFDU(1, 1) = -UX(1)
      DFDUX(1, 1) = -U(1)
      DFDUXX(1, 1) = COEFF
C
      RETURN
      END
C-----
      SUBROUTINE BNDXA(T, U, UX, BVAL, NPDE)

```

```
C-----  
C PURPOSE:  
C     THE SUBROUTINE IS USED TO DEFINE THE BOUNDARY CONDITIONS AT THE  
C     LEFT SPATIAL END POINT X = XA.  
C           B(T, U, UX) = 0  
C  
C-----  
C SUBROUTINE PARAMETERS:  
C INPUT:  
C     INTEGER                NPDE  
C     THE NUMBER OF PDES IN THE SYSTEM.  
C  
C     DOUBLE PRECISION      T  
C     THE CURRENT TIME COORDINATE.  
C  
C     DOUBLE PRECISION      U(NPDE)  
C     U(1:NPDE) IS THE APPROXIMATION OF THE  
C     SOLUTION AT THE POINT (T,XA).  
C  
C     DOUBLE PRECISION      UX(NPDE)  
C     UX(1:NPDE) IS THE APPROXIMATION OF THE  
C     FIRST SPATIAL DERIVATIVE OF THE SOLUTION  
C     AT THE POINT (T,XA).  
C  
C OUTPUT:  
C     DOUBLE PRECISION      BVAL(NPDE)  
C     BVAL(1:NPDE) IS THE BOUNDARY CONTIDITION  
C     AT THE LEFT BOUNDARY POINT.  
C-----
```

DOUBLE PRECISION COEFF

COMMON /BURGER/ COEFF

C-----
 BVAL(1)=U(1)-0.5D0+0.5D0*TANH((-0.5D0*T-0.25D0)/(4.0D0*COEFF))

C

RETURN

END

C-----
 SUBROUTINE BNDXB(T, U, UX, BVAL, NPDE)

C-----

C PURPOSE:

C THE SUBROUTINE IS USED TO DEFINE THE BOUNDARY CONDITIONS AT THE
 C RIGHT SPATIAL END POINT X = XB.

C B(T, U, UX) = 0

C

C-----

C SUBROUTINE PARAMETERS:

C INPUT:

INTEGER

NPDE

C

THE NUMBER OF PDES IN THE SYSTEM.

C

DOUBLE PRECISION

T

C

THE CURRENT TIME COORDINATE.

C

DOUBLE PRECISION

U(NPDE)

C

U(1:NPDE) IS THE APPROXIMATION OF THE
 SOLUTION AT THE POINT (T,XB).

C

C

DOUBLE PRECISION

UX(NPDE)

C UX(1:NPDE) IS THE APPROXIMATION OF THE
 C FIRST SPATIAL DERIVATIVE OF THE SOLUTION
 C AT THE POINT (T,XB).

C

C OUTPUT:

 DOUBLE PRECISION BVAL(NPDE)

C BVAL(1:NPDE) IS THE BOUNDARY CONTIDITION
 C AT THE RIGHT BOUNDARY POINT.

C-----

 DOUBLE PRECISION COEFF

 COMMON /BURGER/ COEFF

C-----

 BVAL(1)=U(1)-0.5D0+0.5D0*TANH((1.D0-0.5D0*T-0.25D0)/(4.0D0*COEFF))

C

 RETURN

 END

C-----

 SUBROUTINE DIFBXA(T, U, UX, DBDU, DBDUX, DBDT, NPDE)

C-----

C PURPOSE:

C THE SUBROUTINE IS USED TO DEFINE THE DIFFERENTIATED BOUNDARY
 C CONDITIONS AT THE LEFT SPATIAL END POINT X = XA. FOR THE
 C BOUNDARY CONDITION EQUATION

C $B(T, U, UX) = 0$

C THE PARTIAL DERIVATIVES DB/DU, DB/DUX, AND DB/DT ARE SUPPLIED
 C BY THIS ROUTINE.

C

C-----

C SUBROUTINE PARAMETERS:

```

C INPUT:
      INTEGER                NPDE
C                            THE NUMBER OF PDES IN THE SYSTEM.
C
      DOUBLE PRECISION      T
C                            THE CURRENT TIME COORDINATE.
C
      DOUBLE PRECISION      U(NPDE)
C                            U(1:NPDE) IS THE APPROXIMATION OF THE
C                            SOLUTION AT THE POINT (T,X).
C
      DOUBLE PRECISION      UX(NPDE)
C                            UX(1:NPDE) IS THE APPROXIMATION OF THE
C                            FIRST SPATIAL DERIVATIVE OF THE SOLUTION
C                            AT THE POINT (T,X).
C OUTPUT:
      DOUBLE PRECISION      DBDU(NPDE, NPDE)
C                            DBDU(I,J) IS THE PARTIAL DERIVATIVE
C                            OF THE I-TH COMPONENT OF THE VECTOR B
C                            WITH RESPECT TO THE J-TH COMPONENT
C                            OF THE UNKNOWN FUNCTION U.
C
      DOUBLE PRECISION      DBDUX(NPDE, NPDE)
C                            DBDUX(I,J) IS THE PARTIAL DERIVATIVE
C                            OF THE I-TH COMPONENT OF THE VECTOR B
C                            WITH RESPECT TO THE J-TH COMPONENT
C                            OF THE SPATIAL DERIVATIVE OF THE
C                            UNKNOWN FUNCTION U.

```



```

C
      DOUBLE PRECISION          DBDT(NPDE)
C                               DBDT(I) IS THE PARTIAL DERIVATIVE
C                               OF THE I-TH COMPONENT OF THE VECTOR B
C                               WITH RESPECT TO TIME T.
      DOUBLE PRECISION COEFF
      COMMON /BURGER/ COEFF
C-----
C-----
C
C   ASSIGN DBDU(1:NPDE,1:NPDE), DBDU(1:NPDE,1:NPDE), AND DBDT(1:NPDE)
C   ACCORDING TO THE RIGHT BOUNDARY CONDITION EQUATION IN TERMS OF
C   U(1:NPDE), UX(1:NPDE), UXX(1:NPDE).
C
      DBDU(1,1) = 1.000
      DBDUX(1,1) = 0.000
      DBDT(1) = -1.00/COEFF/SINH(-(0.500*T+0.2500)/(4.00*COEFF))**2
C
      RETURN
      END
C-----
      SUBROUTINE DIFBXB(T, U, UX, DBDU, DBDUX, DBDT, NPDE)
C-----
C PURPOSE:
C   THE SUBROUTINE IS USED TO DEFINE THE DIFFERENTIATED BOUNDARY
C   CONDITIONS AT THE RIGHT SPATIAL END POINT 1 = XB. FOR THE
C   BOUNDARY CONDITION EQUATION
C                               B(T, U, UX) = 0

```

C THE PARTIAL DERIVATIVES DB/DU, DB/DUX, AND DB/DT ARE SUPPLIED
 C BY THIS ROUTINE.

C

C-----

C SUBROUTINE PARAMETERS:

C INPUT:

INTEGER

NPDE

C

THE NUMBER OF PDES IN THE SYSTEM.

C

DOUBLE PRECISION

T

C

THE CURRENT TIME COORDINATE.

C

DOUBLE PRECISION

U(NPDE)

C

U(1:NPDE) IS THE APPROXIMATION OF THE
 SOLUTION AT THE POINT (T,X).

C

C

DOUBLE PRECISION

UX(NPDE)

C

UX(1:NPDE) IS THE APPROXIMATION OF THE
 FIRST SPATIAL DERIVATIVE OF THE SOLUTION
 AT THE POINT (T,X).

C

C

C

C OUTPUT:

DOUBLE PRECISION

DBDU(NPDE, NPDE)

C

DBDU(I, J) IS THE PARTIAL DERIVATIVE
 OF THE I-TH COMPONENT OF THE VECTOR B
 WITH RESPECT TO THE J-TH COMPONENT
 OF THE UNKNOWN FUNCTION U.

C

C

C

C

DOUBLE PRECISION

DBDUX(NPDE, NPDE)

```

C           DBDUX(I,J) IS THE PARTIAL DERIVATIVE
C           OF THE I-TH COMPONENT OF THE VECTOR B
C           WITH RESPECT TO THE J-TH COMPONENT
C           OF THE FIRST SPATIAL DERIVATIVE OF THE
C           UNKNOWN FUNCTION U.
C
C           DOUBLE PRECISION           DBDT(NPDE)
C           DBDT(I) IS THE PARTIAL DERIVATIVE
C           OF THE I-TH COMPONENT OF THE VECTOR B
C           WITH RESPECT TO TIME T.
C
C           DOUBLE PRECISION COEFF
C           COMMON /BURGER/ COEFF
C-----
C-----
C
C           ASSIGN DBDU(1:NPDE,1:NPDE), DBDU(1:NPDE,1:NPDE), AND DBDT(1:NPDE)
C           ACCORDING TO THE RIGHT BOUNDARY CONDITION EQUATION IN TERMS OF
C           U(1:NPDE), UX(1:NPDE), UXX(1:NPDE).
C
C           DBDU(1,1) = 1.0D0
C           DBDUX(1,1) = 0.0D0
C           DBDT(1) = -1.0D0/COEFF/SINH(-(0.5D0*T-0.75D0)/(4.0D0*COEFF))**2
C
C           RETURN
C           END
C-----
C           SUBROUTINE UINIT(X, U, NPDE)
C-----

```

C PURPOSE:

C THIS SUBROUTINE IS USED TO RETURN THE NPDE-VECTOR OF INITIAL
C CONDITIONS OF THE UNKNOWN FUNCTION AT THE INITIAL TIME T = TO
C AT THE SPATIAL COORDINATE X.

C

C-----

C SUBROUTINE PARAMETERS:

C INPUT:

DOUBLE PRECISION X

C THE SPATIAL COORDINATE.

C

INTEGER NPDE

C THE NUMBER OF PDES IN THE SYSTEM.

C

C OUTPUT:

DOUBLE PRECISION U(NPDE)

C U(1:NPDE) IS VECTOR OF INITIAL VALUES OF
C THE UNKNOWN FUNCTION AT T = TO AND THE
C GIVEN VALUE OF X.

C-----

DOUBLE PRECISION COEFF
COMMON /BURGER/ COEFF

C-----

C

C ASSIGN U(1:NPDE) THE INITIAL VALUES OF U(TO,X).

C

U(1) = 0.5D0-0.5D0*TANH((X-0.25D0)/(4.0D0*COEFF))

C

RETURN

END

The subroutine which represents the exact solution is given as follows.

```

C-----
      SUBROUTINE TRUU(T, X, U, NPDE)
C-----
C  PURPOSE:
C    THIS FUNCTION PROVIDES THE EXACT SOLUTION OF THE PDE.
C-----
C  SUBROUTINE PARAMETERS:
C  INPUT:
      INTEGER                NPDE
C                          THE NUMBER OF PDES IN THE SYSTEM.
C
      DOUBLE PRECISION      T
C                          THE CURRENT TIME COORDINATE.
C
      DOUBLE PRECISION      X
C                          THE CURRENT SPATIAL COORDINATE.
C
C  OUTPUT:
      DOUBLE PRECISION      U(NPDE)
C                          U(1:NPDE) IS THE EXACT SOLUTION AT THE
C                          POINT (T,X).
C-----
      DOUBLE PRECISION COEFF
      COMMON /BURGER/ COEFF

C-----

      U(1) = 0.5D0-0.5D0*TANH((X-0.5D0*T-0.25D0)/(4.0D0*COEFF))

```

C

RETURN

END

4.4 Structure of BACOL

The way in which all the subroutines in the BACOL software work together is illustrated in Figure 4.1. We let MAIN denote the driver program; it is inside the rectangle constructed with dashed lines. Any subroutines inside a single rectangle constructed with solid lines represent software developed by others. Any subroutines inside double rectangles constructed with solid lines represent newly developed software. The notation, $A \rightarrow B$, means subroutine A calls subroutine B .

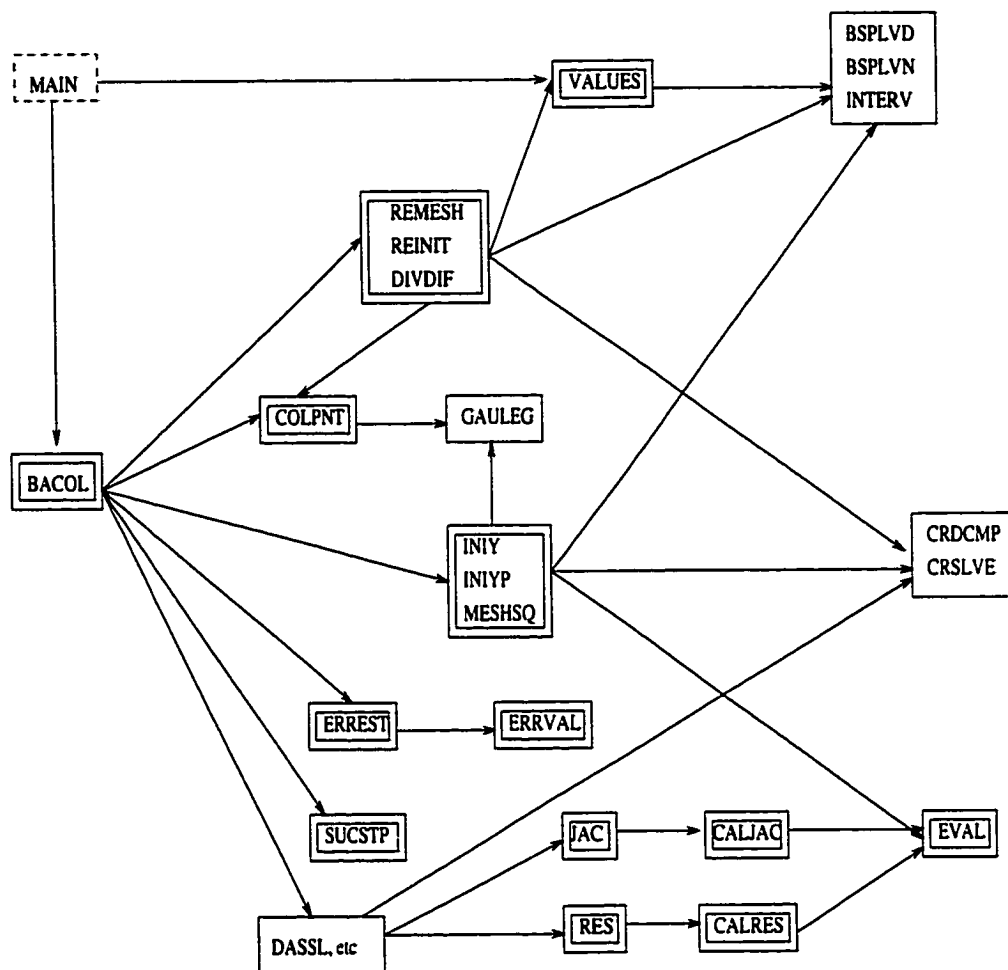


Figure 4.1: Structure of BACOL software

An interesting analysis involves examining the percentage of computer time associated with each subroutine. Obviously, during a typical run, if we can decrease the running time of the subroutines which represent a large percentage of the computer time, the performance of the entire software package can be improved. On the other hand, we can pay less attention to the subroutines which represent a small percentage of the total computer time. This analysis can therefore highlight the components of the software where future modifications would be worthwhile.

The first typical run we consider is based on Problem 1 in Chapter 5 with $\epsilon = 10^{-4}$. We consider a piecewise cubic polynomial for representation of the spatial dependence of our approximate solution. The absolute and relative tolerances are both chosen to be 10^{-6} , and the initial mesh is chosen as a uniform mesh with the number of subintervals equal to 10. All the numerical experiments discussed in this chapter were done on a SUN SPARC station for which the *CPU* clock rate is 480.0 MHz and the main memory clock rate is 96.0 MHz. SUNWspro/bin/f77 is the Fortran compiler. The unix command, *gprof*, is used to obtain the *CPU* time results for each subroutine. Our compiling command is "f77 -pg -u -O ", where "-pg" asks the compiler to prepare for profiling with *gprof*, "-u" asks the compiler to report undeclared variables, "-O" requires a basic level of optimization. In Section 3.2.8 we described modifications we made to DDANRM. We now show a comparison between the results before and after these modifications. Figure 4.2 shows the top 10 subroutines in terms of *CPU* time percentage plus CRDCMP (the 12th place) and CALJAC (the 15th place) before modifications. Figure 4.3 shows the subroutines having the top 10 *CPU* time packages plus CRDCMP (the 11th place) and CALJAC (the 16th place) after modifications. We note that all the subroutines in Figure 4.2 also appear in Figure 4.3.

We observe that before the modifications, the overall *CPU* time is 24.92 seconds and DDANRM takes 5.01 seconds and 20.1% of the overall time; after the modifications, the overall *CPU* time is 23.40 seconds and DDANRM takes 3.52 seconds and 15.0% of the overall costs. As expected, the modifications discussed in Section 3.2.8 save roughly 25% in DDANRM. Since DDANRM uses the largest percentage of *CPU* time, this modification improves the speed of the entire package by 5%.

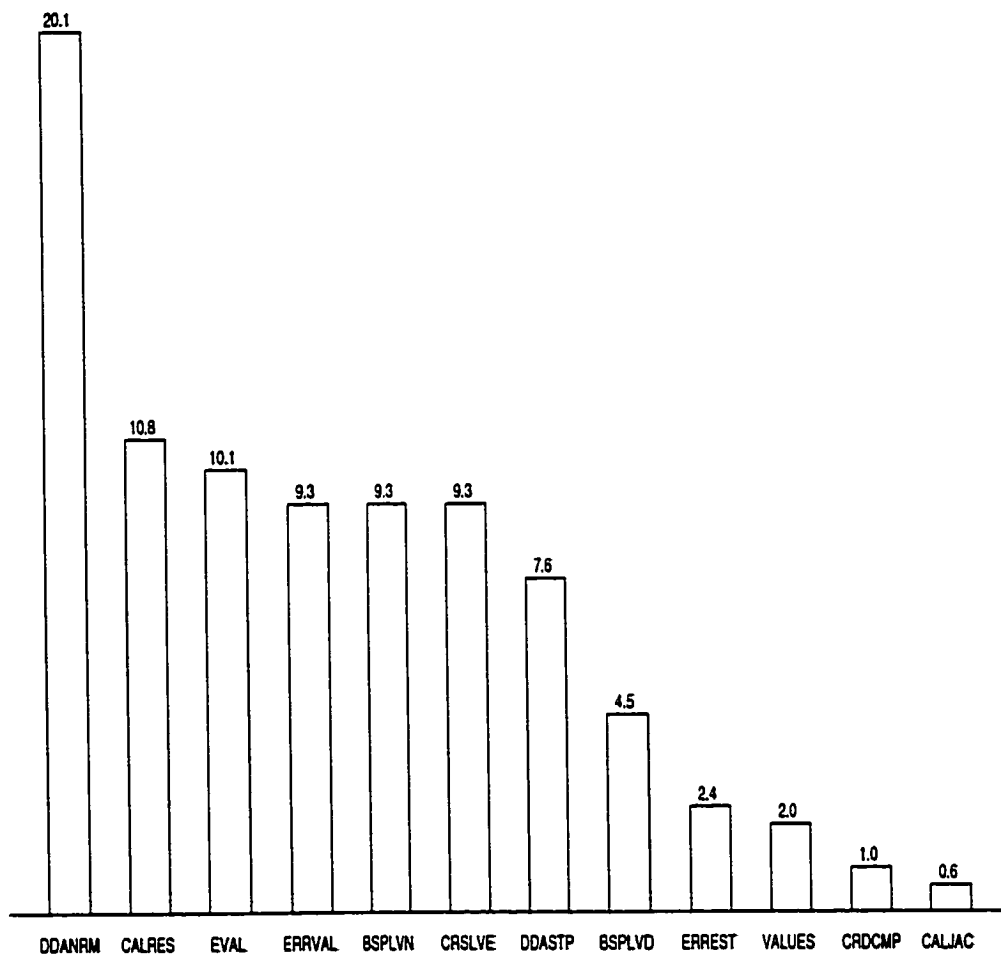


Figure 4.2: Percentage of the *CPU* time in main subroutines: single equation and before modifications to DDANRM.

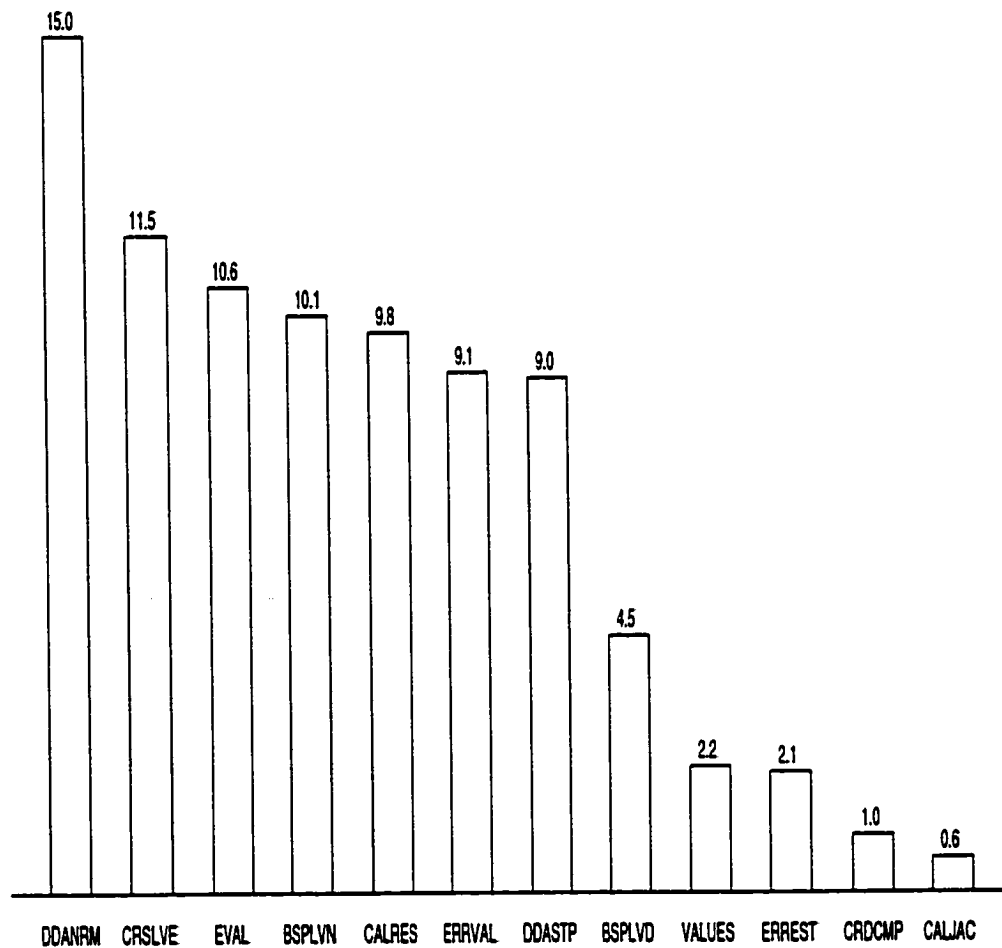


Figure 4.3: Percentage of the *CPU* time in main subroutines: single equation and after modifications to DDANRM.

The second problem we consider is a system of 4 PDEs. By taking 4 copies of Problem 1, we generate a system of PDES whose solution has the same spatial properties as the original problem. We use the same parameter and input data. In Figures 4.4 and 4.5, we show the subroutines having the top 10 *CPU* time packages plus *VALUES* and *BSPLVD*. We note that the same subroutines appear in Figure 4.2 to Figure 4.5. Figure 4.4 shows the results before the modifications to *DDANRM* while Figure 4.5 shows the results after the modifications.

We observe that before the modifications, the overall *CPU* time is 88.75 seconds and *DDANRM* takes 20.89 seconds and 23.5% of the overall time; after the modifications, the overall *CPU* time is 83.65 seconds and *DDANRM* takes 14.00 seconds and 16.7% of the overall time. Once again, we see that the modifications save roughly 25% in *DDANRM*. Another importance observation is that, compared to the single equation case, *BACOL* spends much more time on the linear system solver, i.e., in *CRSLVE* and *CRDCMP*. Comparing Figure 4.2 and Figure 4.4, we see that the *CPU* time percentage for *CRSLVE* increases from 9.3 to 21.6 and the percentage for *CRDCMP* increases from 1.0 to 5.5. The comparison between Figure 4.3 and Figure 4.5 gives the same observation. This is understandable since the cost of a Newton iteration is $\frac{1}{3} * N * NPDE^3 * (p + 1)^3$. Thus when $NPDE = 4$, this cost increases by a factor of 64, which makes *CRSLVE* and *CRDCMP* represent larger percentages of the overall cost.

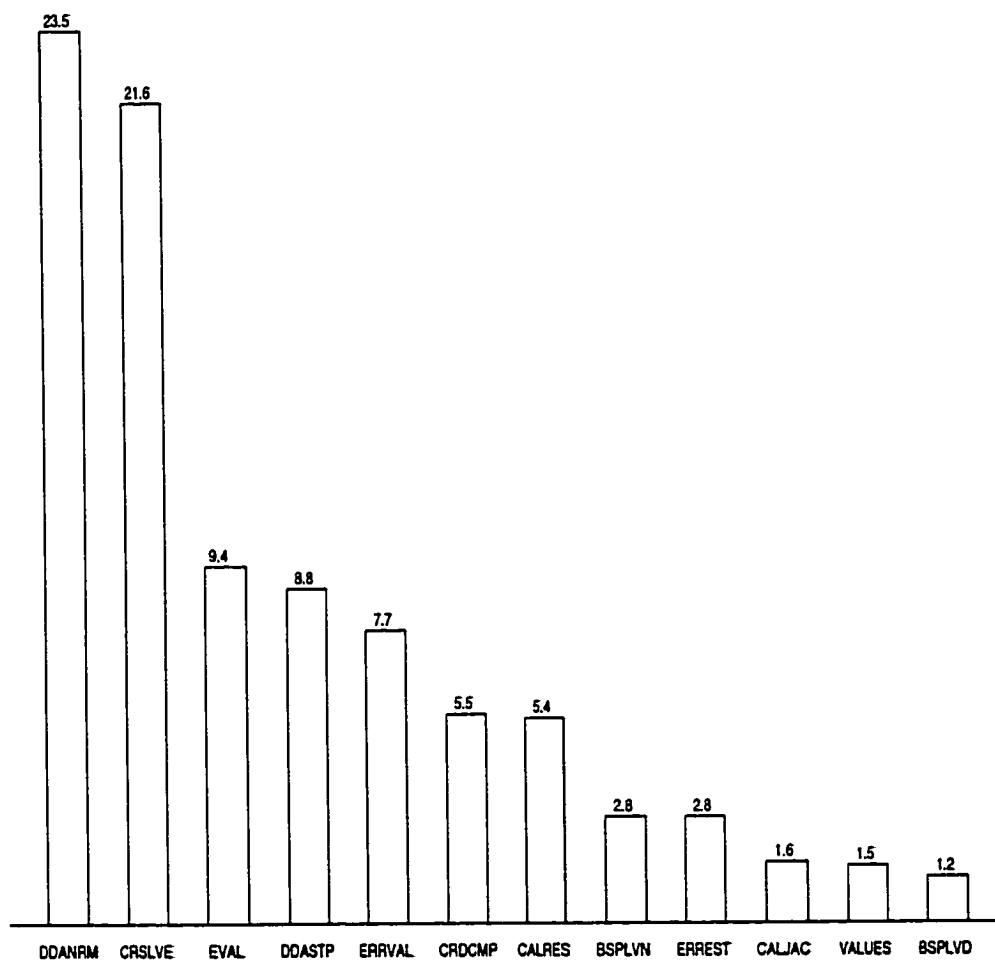


Figure 4.4: Percentage of the *CPU* time in main subroutines: 4 equations and before modifications to DDANRM.

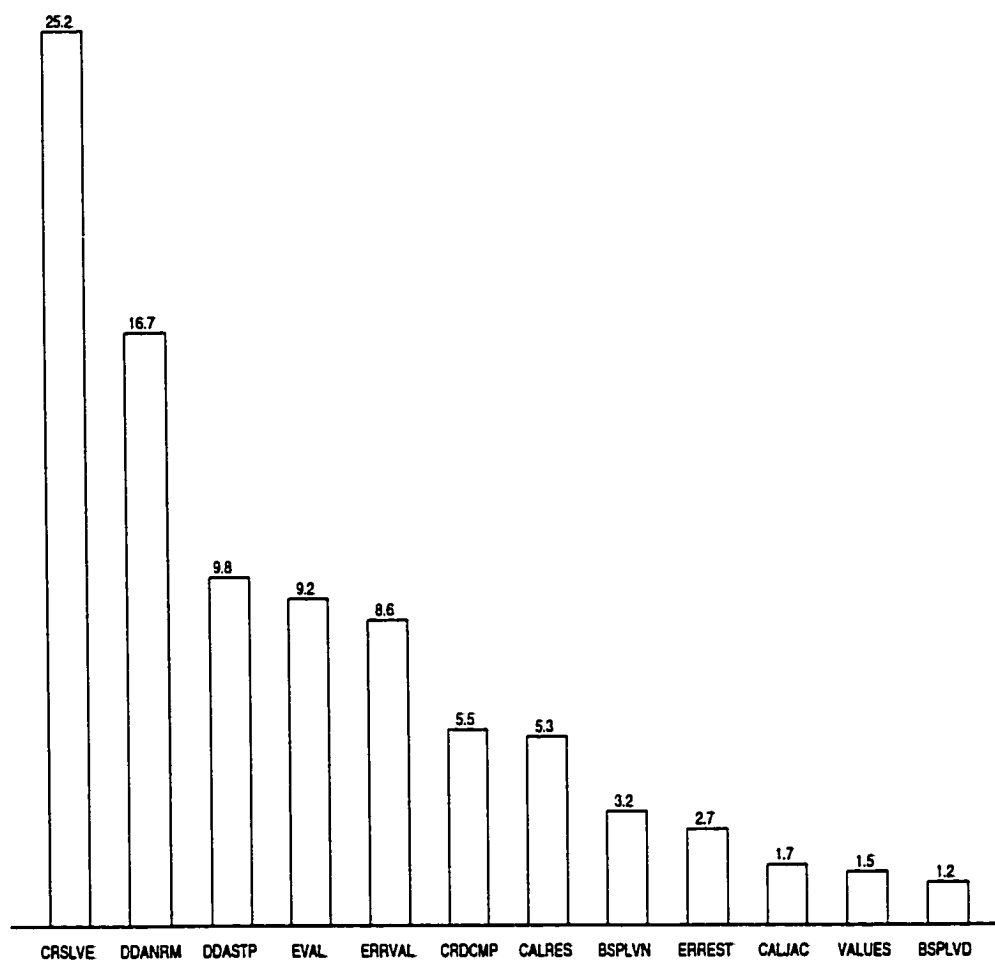


Figure 4.5: Percentage of the *CPU* time in main subroutines: 4 equations and after modifications to DDANRM.

Chapter 5

Numerical Experiments

In this chapter we will consider seven different timer-dependent one-dimensional (i.e., one space dimension) parabolic PDE problems, two of which are systems. A brief introduction to these test problems will be given in Section 5.1. We will then present in Section 5.2 the results of using BACOL to solve one of the test problems and illustrate some of the properties of the code and the effect of the use of different degree of the piecewise polynomials. In Section 5.3 we compare BACOL with several other software packages. We concentrate mostly on the efficiency, i.e., the amount of computer time to achieve a fixed amount of accuracy. In Section 5.4 we investigate the effect of differentiation of the boundary conditions by considering Problem 5, whose boundary conditions are inconsistent with the initial conditions. In Section 5.5 we show that BACOL is able to handle a problem with blow-up. In Section 5.6 we will present a study of the convergence rates of the errors associated with the BACOL computations, based on uniform meshes. The results confirm the expected rate of convergence at non-mesh points and demonstrate superconvergence at the mesh points.

In this chapter, all the numerical experiments are done on a SUN SPARC station, with a *CPU* clock rate of 480.0 MHz and a main memory clock rate of 96.0 MHz. SUN-Wspro/bin/f77 is our Fortran compiler for all the software packages except HPNEW,

for which SUNWspro/bin/f90 is used. Compilation is done using the -O switch.

The notation used and the statistics collected include:

- NINT*: the number of subintervals;
- KCOL*: the number of collocation points per subinterval, which is equal to the degree of the piecewise polynomial minus one, i.e., $p - 1$.
- ATOL*: the scalar absolute tolerance;
- RTOL*: the scalar relative tolerance;
- T_{out}*: the output time;
- X3dM*: the grid dimension along the x-axis for plotting in a 3-D Figure;
- T3dM*: the grid dimension along the t-axis for plotting in a 3-D Figure;
- TOL*: if $ATOL = RTOL$, *TOL* will be used for convenience;
- CPU*: the execution time in seconds;
- NS*: the total number of successful time steps;
- NR*: the total number of times BACOL performs a remeshing.

5.1 Statement of Test Problems

The following problems have been used by many authors for the evaluation of software for the numerical solution of one dimensional time dependent parabolic PDE systems. Some problems include one or more parameters which can be varied to adjust the difficulty of the problem. We now present our test problems.

5.1.1 Problem 1

This is Burgers' equation:

$$u_t = -uu_x + \epsilon u_{xx}, \quad 0 < x < 1, \quad t > 0, \quad (5.1)$$

with initial condition

$$u(x, 0) = \frac{0.1e^{-A_0} + 0.5e^{-B_0} + e^{-C_0}}{e^{-A_0} + e^{-B_0} + e^{-C_0}}, \quad 0 \leq x \leq 1,$$

and boundary conditions

$$u(0, t) = \frac{0.1e^{-A_L} + 0.5e^{-B_L} + e^{-C_L}}{e^{-A_L} + e^{-B_L} + e^{-C_L}}, \quad t \geq 0,$$

$$u(1, t) = \frac{0.1e^{-A_R} + 0.5e^{-B_R} + e^{-C_R}}{e^{-A_R} + e^{-B_R} + e^{-C_R}}, \quad t \geq 0,$$

where

$$A_0 = \frac{0.05}{\epsilon}(x - 0.5), \quad B_0 = \frac{0.25}{\epsilon}(x - 0.5), \quad C_0 = \frac{0.5}{\epsilon}(x - 0.375),$$

$$A_L = \frac{0.05}{\epsilon}(-0.5 + 4.95t), \quad B_L = \frac{0.25}{\epsilon}(-0.5 + 0.75t), \quad C_L = \frac{0.5}{\epsilon}(-0.375),$$

$$A_R = \frac{0.05}{\epsilon}(0.5 + 4.95t), \quad B_R = \frac{0.25}{\epsilon}(0.5 + 0.75t), \quad C_R = \frac{0.5}{\epsilon}(0.625),$$

where, for the viscosity parameter, ϵ , we will consider two values, $\epsilon = 10^{-3}$ and $\epsilon = 10^{-4}$.

This problem is taken from [1, 18], and has the exact solution

$$u(x, t) = \frac{0.1e^{-A} + 0.5e^{-B} + e^{-C}}{e^{-A} + e^{-B} + e^{-C}},$$

where

$$A = \frac{0.05}{\epsilon}(x - 0.5 + 4.95t), \quad B = \frac{0.25}{\epsilon}(x - 0.5 + 0.75t), \quad C = \frac{0.5}{\epsilon}(x - 0.375).$$

In our numerical experiments, we scale the denominator and the numerator to avoid overflow. That is, if $D = \min(A, B, C)$, then we will multiply both the denominator and the nominator by e^D . Therefore, the exact solution becomes

$$u(x, t) = \frac{0.1e^{D-A} + 0.5e^{D-B} + e^{D-C}}{e^{D-A} + e^{D-B} + e^{D-C}}.$$

To avoid underflow, we check $D - A$, $D - B$, and $D - C$. If any of them is less than -35 , we will let the corresponding exponential function be zero (since we use double precision).

The exact solution is plotted in Figure 5.1 for $\epsilon = 10^{-4}$, $0 \leq t \leq 1$, $0 \leq x \leq 1$, $X3dM = 20$, and $T3dM = 20$.

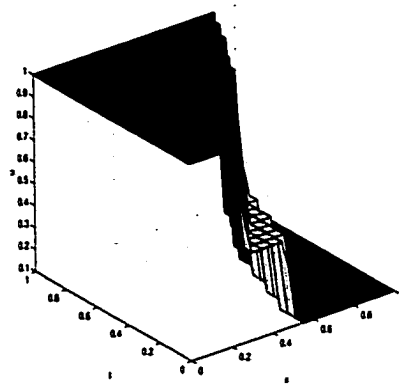


Figure 5.1: Problem 1 for $\epsilon = 10^{-4}$.

The exact solution begins with two wavefronts. They move from left to right and merge to form one wavefront toward the end of computation. As mentioned in [55], the thickness of the wavefronts is $O(\epsilon)$.

5.1.2 Problem 2

Our second problem is also Burgers' equation, [18], but with a different set of initial and boundary conditions. We have

$$u_t = -uu_x + \epsilon u_{xx}, \quad 0 < x < 1, \quad t > 0, \quad (5.2)$$

with initial condition

$$u(x, 0) = 0.5 - 0.5 \tanh\left(\frac{1}{4\epsilon}(x - 0.25)\right), \quad 0 \leq x \leq 1,$$

and boundary conditions

$$u(0, t) = 0.5 - 0.5 \tanh\left(\frac{1}{4\epsilon}(-t - 0.25)\right), \quad t \geq 0,$$

$$u(1, t) = 0.5 - 0.5 \tanh\left(\frac{1}{4\epsilon}(0.75 - t)\right), \quad t \geq 0.$$

The exact solution is

$$u(x, t) = 0.5 - 0.5 \tanh\left(\frac{1}{4\epsilon}(x - t - 0.25)\right).$$

In our numerical experiments, we consider $\epsilon = 10^{-3}$ and $\epsilon = 10^{-4}$.

The exact solution is plotted in Figure 5.2 for $\epsilon = 10^{-4}$, $0 \leq t \leq 1$, $0 \leq x \leq 1$, $X3dM = 20$, and $T3dM = 20$.

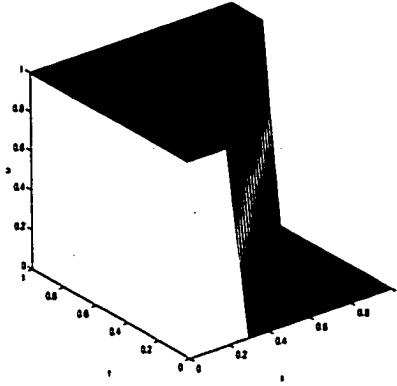


Figure 5.2: Problem 2 for $\epsilon = 10^{-4}$.

We note that the solution in Problem 2 has only one wavefront whose thickness is $O(\epsilon)$.

5.1.3 Problem 3

The third problem is taken from [26]; it has the form

$$u_t = u_{xx} + u^p, \quad 0 < x < 1, \quad t > 0, \quad (5.3)$$

with initial condition

$$u(x, 0) = 20 \sin(\pi x), \quad 0 \leq x \leq 1,$$

and boundary conditions

$$u(0, t) = 0, \quad u(1, t) = 0, \quad t \geq 0.$$

We only consider the case when $p = 2$. After some finite time the exact solution becomes unbounded; See [26] for further discussion. We note that for such cases,

it is essential that the numerical solution preserves this property; i.e., the numerical solution should also blow up at a time that is close to that at which the exact solution does.

The exact solution is not available; however, a high-precision numerical solution obtained using BACOL is shown in Figure 5.3. With $ATOL = RTOL = 10^{-7}$, $KCOL = 5$, and $NINT = 5$ as the initial inputs, we found that the blow-up time was approximately 0.082437272703, when the maximum solution was around 5×10^{12} . We stopped BACOL at that time and plotted the solution for $0 \leq x \leq 1$ and $0 \leq t \leq 0.082437272703$. Since the solution keeps increasing and the maximum value is obtained at $x = 0.5$, we divided the solution by $U(0.5, t)$, which makes the maximum value 1 all times. In Figure 5.3, we set $X3dM = 100$ and $T3dM = 100$, and we plot $U(x, t)/U(0.5, t)$.

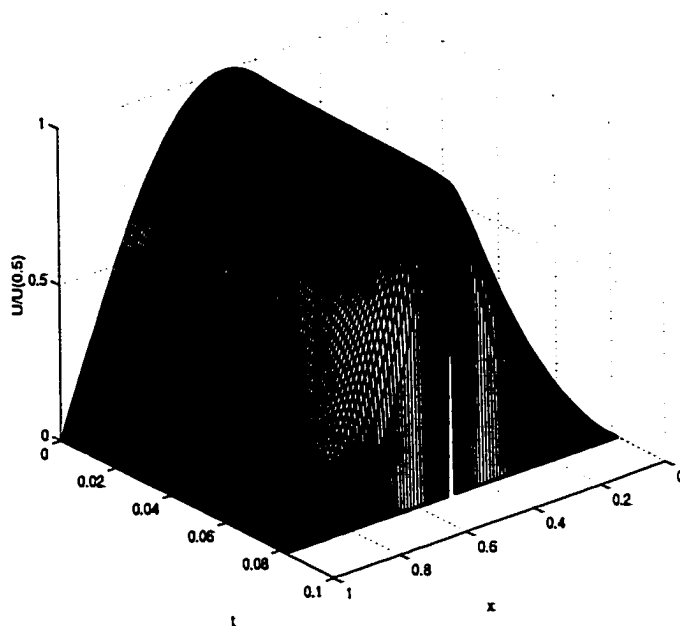


Figure 5.3: Problem 3 for $\epsilon = 10^{-6}$.

5.1.4 Problem 4

The fourth problem is the Cahn-Allen equation [58], which has the form

$$u_t = \epsilon u_{xx} - u^3 + u, \quad 0 < x < 1, \quad t > 0, \quad (5.4)$$

with initial condition

$$u(x, 0) = 0.01 \cos(10\pi x), \quad 0 \leq x \leq 1,$$

and boundary conditions

$$u_x(0, t) = 0, \quad u_x(1, t) = 0, \quad t \geq 0.$$

For this problem we choose $\epsilon = 10^{-6}$.

No exact solution is available; a high-precision numerical solution obtained using BACOL is shown in Figure 5.4. We have $ATOL = 10^{-6}$, $RTOL = 0$ with the initial input $KCOL = 5$ and $NINT = 5$. The solution is plotted for $0 \leq x \leq 1$ and $0 \leq t \leq 8$ and we set $X3dM = 40$ and $T3dM = 40$.

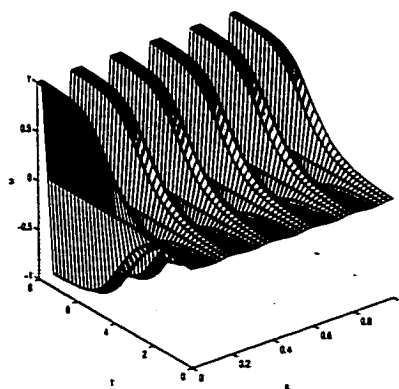


Figure 5.4: Problem 4 for $\epsilon = 10^{-6}$.

There are two phases of solution behavior. First, the solution quickly generates sharp interfaces and is close to a step function, as shown in Figure 5.5

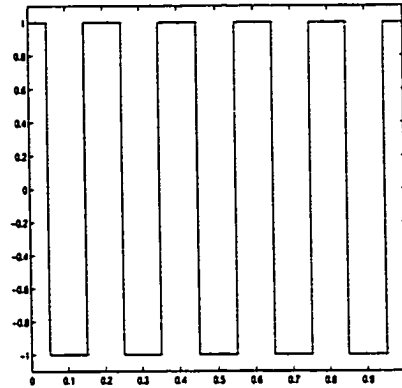


Figure 5.5: The step function.

In the second phase, the solution becomes nearly constant.

5.1.5 Problem 5

The fifth problem is the Brusselator problem with diffusion [58]; it has the form

$$\begin{aligned} u_t &= 1 + u^2v - 4.4u + \epsilon u_{xx}, & 0 < x < 1, & \quad t > 0, \\ v_t &= 3.4u - u^2v + \epsilon v_{xx}, & 0 < x < 1, & \quad t > 0, \end{aligned} \quad (5.5)$$

with initial conditions

$$\begin{aligned} u(x, 0) &= 0.5, & 0 \leq x \leq 1, \\ v(x, 0) &= 1 + 5x + 0.25 \tanh(20x) - 0.25 \tanh(20(x - 1)), & 0 \leq x \leq 1, \end{aligned}$$

and boundary conditions

$$u_x(0, t) = v_x(0, t) = u_x(1, t) = v_x(1, t) = 0, \quad t \geq 0.$$

We consider $\epsilon = 2 \times 10^{-5}$.

This is an interesting problem since the initial conditions are inconsistent with the boundary conditions, i.e., the initial condition given above for $v(t, 0)$ implies that

$v_x(0,0) = 5(2 - \frac{1}{\cosh^2(20)}) \approx 10$, which is inconsistent with the boundary condition $v_x(0,0) = 0$.

The problem does not have an exact solution; high-precision numerical approximations for u and v are shown in Figures 5.6 and 5.7, which are obtained using BACOL. We have $ATOL = RTOL = 10^{-6}$ with the initial input $KCOL = 5$ and $NINT = 5$. The solution is plotted for $0 \leq x \leq 1$ and $0 \leq t \leq 35.83$, and we set $X3dM = 100$ and $T3dM = 1000$.

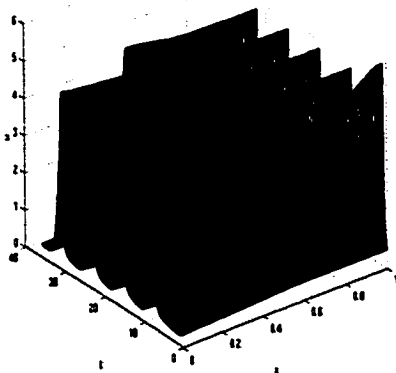
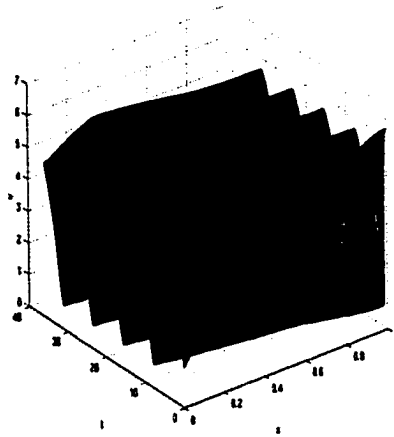


Figure 5.6: Problem 5 for $u(x, t)$.

Figure 5.7: Problem 5 for $v(x, t)$.

The solutions begin with two smooth initial conditions (constant initial values for $u(x, 0)$ and almost linear for $v(x, 0)$). A wavefront in each solution is immediately generated at the right boundary, which then moves from right to left. After the wavefront moves out of the left boundary, another wavefront appears at the right boundary and the process is repeated.

5.1.6 Problem 6

The sixth problem is also taken from [58], where the author considers the reaction-diffusion-convection system for modeling a catalytic surface reaction. It has the form

$$\begin{aligned}
 (u_1)_t &= -(u_1)_x + n(D_1 u_3 - A_1 u_1(1 - u_3 - u_4)) + \frac{1}{Pe_1}(u_1)_{xx}, & 0 < x < 1, & \quad t > 0, \\
 (u_2)_t &= -(u_2)_x + n(D_2 u_4 - A_2 u_2(1 - u_3 - u_4)) + \frac{1}{Pe_1}(u_2)_{xx}, & 0 < x < 1, & \quad t > 0, \\
 (u_3)_t &= A_1 u_1(1 - u_3 - u_4) - D_1 u_3 - R u_3 u_4(1 - u_3 - u_4)^2 + \frac{1}{Pe_2}(u_3)_{xx}, & 0 < x < 1, & \quad t > 0, \\
 (u_4)_t &= A_2 u_2(1 - u_3 - u_4) - D_2 u_4 - R u_3 u_4(1 - u_3 - u_4)^2 + \frac{1}{Pe_2}(u_4)_{xx}, & 0 < x < 1, & \quad t > 0,
 \end{aligned}
 \tag{5.6}$$

with initial conditions

$$u_1(x, 0) = 2 - r, \quad u_2(x, 0) = r, \quad u_3(x, 0) = u_4(x, 0) = 0, \quad 0 < x < 1,$$

and boundary conditions

$$\frac{1}{Pe_1}(u_1)_x(0, t) = -(2 - r - u_1), \quad \frac{1}{Pe_1}(u_2)_x(0, t) = -(r - u_2), \quad t > 0,$$

$$(u_3)_x(0, t) = (u_4)_x(0, t) = 0, \quad t > 0,$$

$$(u_1)_x(1, t) = (u_2)_x(1, t) = (u_3)_x(1, t) = (u_4)_x(1, t) = 0, \quad t > 0,$$

where u_1 and u_2 are nondimensionalized concentrations, u_3 and u_4 are coverages of adsorbed reactants on the catalytic wall, Pe_1 and Pe_2 are Peclet numbers, and D_1 , D_2 , R , A_1 and A_2 are Damkohler numbers. This problem includes diffusion, reaction and convection. We choose $A_1 = A_2 = 30$, $D_1 = 1.5$, $D_2 = 1.2$, $R = 1000$, $r = 0.96$, $n = 1$ and $Pe_1 = Pe_2 = 100$.

The problem does not have an exact solution; high-precision numerical approximations for the solution components are shown in Figures 5.8-5.11, which are obtained using BACOL. We have $ATOL = RTOL = 10^{-5}$ with the initial input $KCOL = 2$ and $NINT = 10$. The solution is plotted for $0 \leq x \leq 1$ and $0 \leq t \leq 18$, and we set $X3dM = 100$ and $T3dM = 100$.

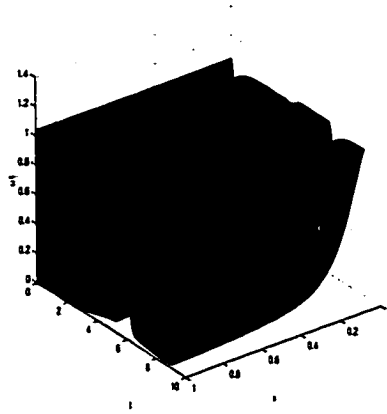


Figure 5.8: Problem 6 for $u_1(x, t)$.

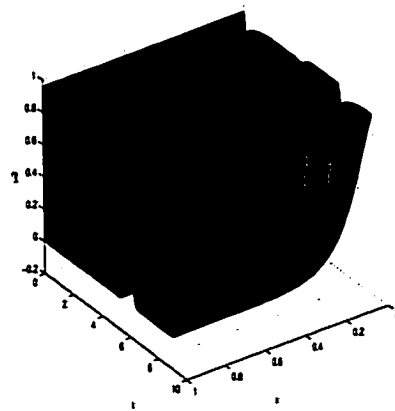


Figure 5.9: Problem 6 for $u_2(x, t)$.

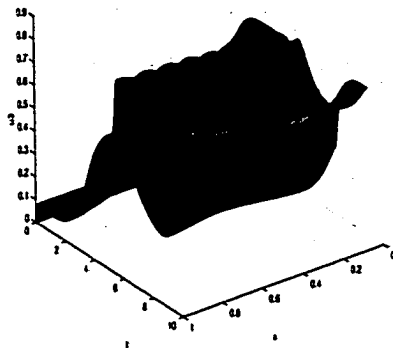


Figure 5.10: Problem 6 for $u_3(x, t)$.

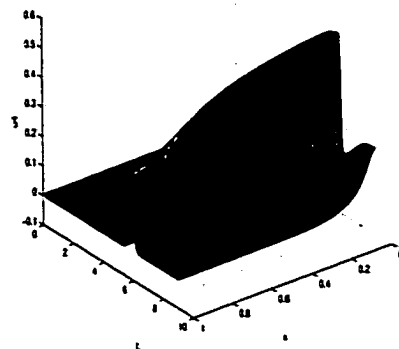


Figure 5.11: Problem 6 for $u_4(x, t)$.

5.1.7 Problem 7

The seventh problem is taken from [28]. This equation contains an exponential non-linear coefficient for the term u_{xx} ; it has the form

$$u_t = (e^{au}u_x)_x, \quad 0 < x < 1, \quad t > 0, \quad (5.7)$$

with initial condition

$$u(x, 0) = bx, \quad 0 \leq x \leq 1,$$

and boundary conditions

$$u(0, t) = 0, \quad u(1, t) = b, \quad t \geq 0.$$

We only consider the case when $a = 5$ and $b = 2$. The steady-state solution is given by

$$u(x, t \rightarrow \infty) = \frac{1}{a} \log[1 + (e^{ab} - 1)x].$$

The exact solution is not available; however, a high-precision numerical solution obtained using BACOL is shown in Figure 5.12. We have $ATOL = RTOL = 10^{-6}$ with the initial input $KCOL = 5$ and $NINT = 5$. The solution is plotted for $0 \leq x \leq 1$ and $0 \leq t \leq 0.5$ and we set $X3dM = 40$ and $T3dM = 40$.

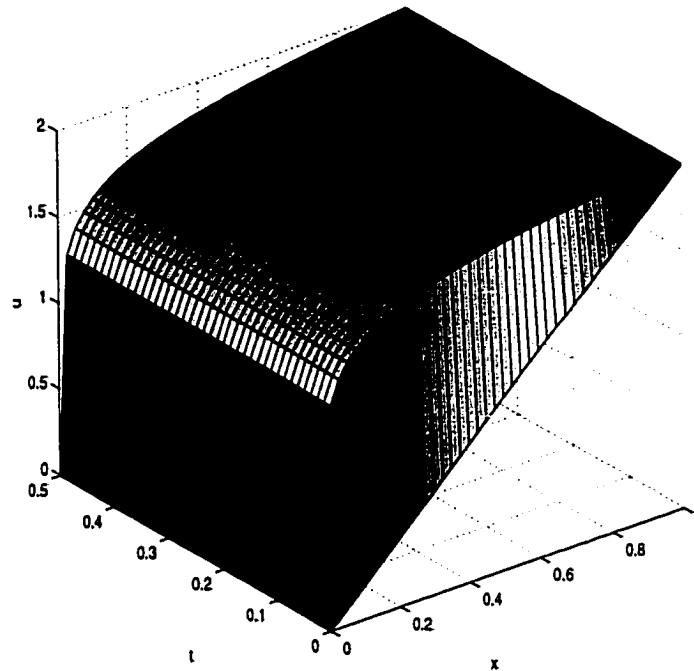


Figure 5.12: Problem 7 for $u(x, t)$.

5.2 Numerical Validation of BACOL

In this section we consider only Problem 1, to which we have applied BACOL with $KCOL = 2, 5, 7, 10$ and $TOL = 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}$. We will choose a uniform mesh with $NINT = 10$ as the initial mesh, $T_{out} = 1$ and let $ATOL = RTOL = TOL$.

5.2.1 TOL and Global Error Proportionality

First we would like to show, for BACOL, the relationship between TOL and the global error. The results are shown in Table 5.1 and 5.2 for $\epsilon = 10^{-3}$ and 10^{-4} , respectively.

<i>TOL</i>	<i>KCOL</i>			
	2	5	7	10
10^{-2}	$1.04 \cdot 10^{-2}$	$8.17 \cdot 10^{-3}$	$3.46 \cdot 10^{-3}$	$1.68 \cdot 10^{-2}$
10^{-4}	$1.04 \cdot 10^{-4}$	$1.16 \cdot 10^{-5}$	$2.18 \cdot 10^{-5}$	$4.82 \cdot 10^{-6}$
10^{-6}	$8.35 \cdot 10^{-7}$	$5.96 \cdot 10^{-7}$	$1.44 \cdot 10^{-7}$	$7.86 \cdot 10^{-8}$
10^{-8}	$5.30 \cdot 10^{-9}$	$8.91 \cdot 10^{-9}$	$9.10 \cdot 10^{-9}$	$1.79 \cdot 10^{-9}$
10^{-10}	$6.18 \cdot 10^{-11}$	$1.02 \cdot 10^{-10}$	$2.45 \cdot 10^{-10}$	$7.90 \cdot 10^{-11}$

Table 5.1: L^2 -norm error for Problem 1 with $\epsilon = 10^{-3}$.

<i>TOL</i>	<i>KCOL</i>			
	2	5	7	10
10^{-2}	$1.05 \cdot 10^{-1}$	$9.05 \cdot 10^{-2}$	$7.44 \cdot 10^{-2}$	$9.99 \cdot 10^{-2}$
10^{-4}	$5.51 \cdot 10^{-4}$	$2.34 \cdot 10^{-4}$	$2.14 \cdot 10^{-4}$	$8.08 \cdot 10^{-5}$
10^{-6}	$4.99 \cdot 10^{-6}$	$9.16 \cdot 10^{-7}$	$1.01 \cdot 10^{-6}$	$1.16 \cdot 10^{-6}$
10^{-8}	$2.56 \cdot 10^{-8}$	$2.29 \cdot 10^{-8}$	$5.59 \cdot 10^{-9}$	$1.27 \cdot 10^{-8}$
10^{-10}	$1.93 \cdot 10^{-10}$	$1.58 \cdot 10^{-10}$	$9.80 \cdot 10^{-11}$	$1.53 \cdot 10^{-10}$

Table 5.2: L^2 -norm error for Problem 1 with $\epsilon = 10^{-4}$.

We see that the global error is close to (or less than) TOL , for all choice of $KCOL$ and TOL . This indicates that our spatial error control strategy, together with our choice of temporal tolerance, works very well.

5.2.2 Relationship between $KCOL$ and CPU time

Second we would like to examine the relationship between $KCOL$ and CPU time.

<i>TOL</i>	<i>KCOL</i>			
	2	5	7	10
10^{-2}	0.175151	0.378189	0.675506	0.977439
10^{-4}	0.850675	1.22791	1.66904	3.20079
10^{-6}	3.52663	3.36868	4.26306	6.26880
10^{-8}	18.8934	10.34185	10.9151	15.3657
10^{-10}	106.305	36.1691	31.9214	33.5394

Table 5.3: *CPU* time for Problem 1 with $\epsilon = 10^{-3}$.

<i>TOL</i>	<i>KCOL</i>			
	2	5	7	10
10^{-2}	1.46985	3.93385	6.40349	10.5109
10^{-4}	9.31914	16.7171	24.6511	40.7753
10^{-6}	34.9236	45.6280	65.6265	102.898
10^{-8}	176.042	118.002	154.531	199.561
10^{-10}	1033.06	385.704	369.964	426.427

Table 5.4: *CPU* time for Problem 1 with $\epsilon = 10^{-4}$.

We observe that when *TOL* is relatively large, BACOL with *KCOL* = 2 is the most efficient. When *TOL* decreases, BACOL with larger *KCOL* values becomes more suitable. For example, when $TOL = 10^{-8}$, BACOL with *KCOL* = 5 takes the least time; and when $TOL = 10^{-10}$, BACOL with *KCOL* = 7 is the most efficient. This can be explained in terms of the number of operations associated with the Newton iterations. For a given *TOL* and a fixed *KCOL*, let $NINT_{KCOL}$ denote the number of subintervals which is necessary to achieve the error requirement. As mentioned before, BACOL employs COLROW as the linear system solver. For each Newton iteration, the number of multiplications required by COLROW to solve the first ABD matrix is roughly $\frac{1}{3} * NINT_{KCOL} * NPDE^3 * (KCOL + 2)^3$. Recall that COLROW is called twice to solve the two decoupled ABD matrices. Therefore, the total number of multiplications for one Newton iteration is approximately $\frac{1}{3} * NINT_{KCOL} * NPDE^3 * ((KCOL + 2)^3 + (KCOL + 3)^3)$. Recall that $KCOL = p - 1$; (3.63) thus becomes,

for a quasiuniform mesh,

$$\|E\| \propto (NINT_{KCOL})^{-(KCOL+2)}. \quad (5.8)$$

For different choices of $KCOL$, e.g. $KCOL = 2$ and $KCOL = 5$, assuming we want to achieve the same error requirement, (5.8) implies that we must have (approximately)

$$(NINT_2)^4 = (NINT_5)^7. \quad (5.9)$$

Therefore, (5.9) suggests that, if the tolerance is decreased, BACOL, in order to control the spatial error, from (5.9) would have to use a much larger $NINT_2$ than $NINT_5$. This explains why BACOL with $KCOL = 5$ needs less *CPU* time than BACOL with $KCOL = 2$ for smaller tolerances.

In general, for a fixed mesh, we expect that if the solution is smooth and high accuracy is required, a high order method will be more efficient than a low order method. Table 5.3 and 5.4 show that, when BACOL is applied to problems with solutions exhibiting narrow wavefronts and higher accuracy is required, use of a higher spatial order is preferred. This suggests an interesting research direction in which we explore how BACOL might be able to automatically estimate a suitable order for the piecewise polynomials during the remeshing phase.

5.2.3 Number of Steps VS. Number of Remeshings

We will examine the relationship between the total number of successful time steps (NS) and the total number of times BACOL performs a remeshing (NR).

<i>TOL</i>	<i>KCOL</i>			
	2	5	7	10
10^{-2}	$\frac{226}{29} \approx 7.8$	$\frac{270}{19} \approx 14.2$	$\frac{396}{20} \approx 19.8$	$\frac{532}{3} \approx 177.3$
10^{-4}	$\frac{838}{79} \approx 10.6$	$\frac{956}{46} \approx 20.8$	$\frac{985}{35} \approx 28.1$	$\frac{1111}{29} \approx 38.3$
10^{-6}	$\frac{1750}{88} \approx 19.9$	$\frac{2230}{74} \approx 30.1$	$\frac{2118}{57} \approx 37.2$	$\frac{2308}{38} \approx 60.7$
10^{-8}	$\frac{3350}{87} \approx 38.5$	$\frac{4806}{101} \approx 47.6$	$\frac{4936}{94} \approx 52.5$	$\frac{5160}{61} \approx 84.6$
10^{-10}	$\frac{6896}{92} \approx 75.0$	$\frac{10216}{116} \approx 88.1$	$\frac{10586}{137} \approx 77.3$	$\frac{10000}{97} \approx 103.1$

Table 5.5: NS/NR for Problem 1 with $\epsilon = 10^{-3}$.

<i>TOL</i>	<i>KCOL</i>			
	2	5	7	10
10^{-2}	$\frac{1472}{139} \approx 10.6$	$\frac{2420}{103} \approx 23.5$	$\frac{2754}{111} \approx 24.8$	$\frac{3137}{47} \approx 66.7$
10^{-4}	$\frac{8800}{779} \approx 11.3$	$\frac{10482}{437} \approx 24.0$	$\frac{11655}{344} \approx 33.9$	$\frac{13248}{275} \approx 48.2$
10^{-6}	$\frac{17838}{971} \approx 18.4$	$\frac{24806}{885} \approx 28.0$	$\frac{37189}{692} \approx 53.7$	$\frac{31398}{508} \approx 61.8$
10^{-8}	$\frac{41199}{992} \approx 41.5$	$\frac{48831}{1042} \approx 46.9$	$\frac{100752}{1157} \approx 87.1$	$\frac{72691}{797} \approx 91.2$
10^{-10}	$\frac{85218}{1036} \approx 82.3$	$\frac{104303}{1198} \approx 87.2$	$\frac{120045}{1317} \approx 91.1$	$\frac{249732}{1117} \approx 223.6$

Table 5.6: NS/NR for Problem 1 with $\epsilon = 10^{-4}$.

From Table 5.5 and 5.6, we observe that, when *TOL* is fixed, BACOL with larger *KCOL* values takes, on average, more time steps between consecutive remeshings; and when *KCOL* is fixed, the number of time steps between consecutive remeshings, on average, increases when *TOL* decreases. Future investigation on such behaviors is required.

In Section 3.2.6, we mention that in order to continue the integration after a remeshing, the worst case arises when DASSL is using a BDF method with order 5. In this case, we need to interpolate the current step and the previous 5 steps. That is, we need to solve six linear systems. If we also reject the step, in the worst case, the cost of one remeshing process is then equal to 7 times the cost of a successful step. Based on this observation, we can estimate the total number of Newton iterations. From Table 5.5 and 5.6, we note that BACOL with larger *KCOL* values performs more

Newton iterations than BACOL with smaller $KCOL$ values.

5.3 Numerical Comparisons of MOL Packages

A major decision when one is assessing software is what performance criteria to use. There are many important criteria, such as ease of use, absence of errors, robustness, storage requirements, flexibility, efficiency, etc. Some of these design criteria may conflict with each other. For instance, providing users with many options may affect the ease of use although it may improve the flexibility; requiring users to provide a subroutine for calculating an analytical Jacobian matrix, which is a common source of programming errors especially when $NPDE$ is large, improves the efficiency but also affects the ease of use. Thus it is clear that the task of comparison of different codes is nontrivial, and a comparison is obviously biased by the class of test problems and comparison criteria. The conclusions can be easily changed if the problems or criteria are changed. However, this is not to say that attempting a comparison is impossible. One can often specify the types of problems for which a code is more suitable. The primary concern of our remeshing strategy is to obtain an approximate solution using as little computer time as possible for a given accuracy. Therefore, *efficiency* will be the major criterion in our code comparison.

In this section, we will compare BACOL with several codes which are able to solve one-dimensional time-dependent parabolic PDE systems. We will only choose $KCOL = 2$ or $KCOL = 5$ for BACOL. We choose all the comparison routines to be double-precision MOL codes since BACOL is double-precision. For all codes we choose the absolute tolerance to be the same as the relative tolerance, and for each experiment, let TOL denote the tolerance. The initial mesh is chosen to be a uniform mesh for all the packages. For all codes except HPNEW and BACOL, the number of subintervals will be chosen a priori to be almost optimal in a sense that will be discussed later in this section. We note that HPNEW has the ability to add or remove mesh points, or vary the degrees of piecewise polynomials in different subintervals. We thus choose the

initial mesh for HPNEW to be uniform with 40 subintervals, and the initial piecewise polynomial is of degree 6 on each subinterval at the initial mesh. The initial mesh for BACOL is chosen to be uniform with 10 subintervals for $KCOL = 2$ and 5. We will now give a brief description of the codes we will use in our comparisons.

- EPDCOL by Keast and Muir [46] is a modification of PDECOL [52, 53]. A collocation method is applied using B-splines as the piecewise polynomial basis. The collocation points in each subinterval are chosen to be Gauss-Legendre points. The degree of piecewise polynomial can vary from 3 to 19. The boundary conditions are differentiated and coupled with the ODE system from the spatial discretizations. GEARIB ([36]) is employed to solve the resulting ODE system. Only a relative tolerance can be imposed. EPDCOL uses a fixed-mesh and thus has no spatial error control. In our numerical experiments we consider $KCOL = 2$ and $KCOL = 5$, which are the same choices as for BACOL. EPDCOL requires the user to supply the initial stepsize for the ODE solver, which is chosen to be 10^{-9} in our experiments.
- D03PPF is in the NAG library. It is able to solve a system of 1-D PDEs coupled with ordinary differential equations. The spatial discretization is done using second-order finite difference methods. The resulting DAE system is integrated using a BDF method or a Theta method which switches between Newton's method and functional iteration. For our calculations we use the BDF methods for time integration and chose the banded linear system solver option. D03PPF is based on SPRINT [13]. It employs an h-refinement approach using a fixed number of mesh points and therefore no spatial error control is available. The user needs to supply a subroutine MONITF which is used as a monitor function for the code to choose a mesh which equally distributes the integral of the monitor function over the domain. In our experiments the monitor function is chosen as the second derivative for a single equation, and $1 + (\sum_{i=1}^{NPDE} |(u_i)_{xx}|)/NPDE$ for a system of $NPDE$ equations. The user also needs to specify how often a remeshing is performed; i.e., after a fixed number

of steps, or after a specified fixed time interval, the code will decide whether a remeshing should be performed. During our testing we require this code to check the remeshing criterion after every 3 time steps.

- TOMS731 is an r-refinement package written by Blom and Zegeling [19]. A finite-element method of second order accuracy is used for the spatial discretization. The time integration is done using DASSL. The monitor function we choose is $\sqrt{0.01 + (\sum_{i=1}^{NPDE} u_x^2)/NPDE}$, which is recommended by the authors. To our understanding, there is no subroutine inside TOMS731 which allows one to compute solution values for non-mesh points. Therefore, after TOMS731 reaches T_{out} , our driver program will call a subroutine from the NAG library, D03PZF, which uses a cubic interpolant to generate solution values. As with all r-refinement codes, the number of mesh points is fixed and spatial error control is therefore not available.
- MOVCOL is an experimental code developed by Huang and Russell [41]. It employs the r-refinement approach. A cubic Hermite collocation approach is used for spatial discretization of the PDE, and a standard central finite difference discretization is used for the mesh equations. DASSL is employed to solve the resulting DAE system. No spatial error control is available.
- HPNEW [59], also an experimental code written in Fortran 90, is a modification of HPDASSL [58] developed by Moore. It applies hp-refinement using a local refinement strategy. A remeshing is performed after every 5 time steps. In the README file, the author claims that HPNEW can deal with problems with mixed boundary conditions. However, from the documentation of the version we obtained from the website (<http://faculty.smu.edu/pmoore/papers.html>), it can only handle Dirichlet and Neumann boundary conditions, and can therefore, not handle problems with mixed boundary conditions, e.g., Problem 6. The inclusion of an h-refinement capability allows this code to control the spatial error.

From the above descriptions, we see that, apart from BACOL, HPNEW is the only package which is able to add or remove mesh points during the computation. Therefore, to obtain “almost optimal” results using EPDCOL, D03PPF, TOMS731, and MOVCOL, we begin with $TOL = 10^{-2}$, and then find a suitable value of $NINT$ as follows. For any given tolerance, we gradually increase $NINT$ and compute solutions and error estimates until the L^2 -norm of the error is close to 2 or 3 times the tolerance (in which case the packages generally work most efficiently), or until the L^2 -norm of the error appears to no longer be dependent on $NINT$ (which implies that the time error is dominating). Then this value of $NINT$ is considered to be “almost optimal” for the given TOL . After that we decrease TOL by a factor of 1/10, and repeat the process again.

During our experiments, we found that EPDCOL, D03PPF and MOVCOL were more consistent in terms of $NINT$ and the L^2 -norm error, i.e., in general, the L^2 -norm error decreases when $NINT$ increases. However, there is some anomolous behaviour shown by TOMS731:

- for a given TOL , most of time there is a value K , such that if $NINT > K$, TOMS731 will fail to converge. That is, TOMS731 seems to have trouble when $NINT$ is large. Apart from that, even for a value of $NINT$ such that $NINT < K$, the L^2 -norm error does not always decrease when $NINT$ increases. For example, when we consider Problem 1 with $\epsilon = 10^{-4}$, the package always fails to converge when $NINT > 100$. And when we choose $TOL = 10^{-3}$ and $NINT = 50$, we obtain the L^2 -norm error, at $T_{out} = 1$, as 6.63×10^{-3} ; however, if we choose $NINT = 70$ with the same tolerance, we obtain the error as 3.26×10^{-2} . Similar behaviour is shown for other values of TOL and $NINT$.

There is also some anomolous behaviour with other codes:

- Sometimes when TOL is relatively large and $NINT$ is not large enough, MOVCOL will fail the error test or the convergence test in the Newton iteration process during the integration. For example, considering Problem 1 with $\epsilon = 10^{-4}$,

if we choose $TOL = 10^{-2}$, $NINT = 40$, the error at $T_{out} = 1$ is obtained as 2.18×10^{-2} and the *CPU* time is 1.39 seconds. However, if we choose $NINT = 20$ or 30 with the same tolerance, MOVCOL fails the error test in DASSL repeatedly. Also, for $NINT = 30$, it takes 113 seconds before reaching T_{out} .

- In general, for a given code, the *CPU* time will increase when TOL decreases. However, sometimes EPDCOL does not act this way. For example, consider Problem 2 with $\epsilon = 10^{-4}$ and apply EPDCOL with $NINT = 4500$. If we choose $TOL = 10^{-4}$, at $T_{out} = 1$ the L^2 -norm error is 1.27×10^{-4} and the *CPU* time is 5943.03 seconds; however, if we choose $TOL = 10^{-5}$, the error is 9.26×10^{-5} and the *CPU* time is only 889.359 seconds. Fortunately, this kind of behaviour is not common.

As mentioned before, efficiency is our main criterion in the comparisons. We thus plot the L^2 -norm error and the total *CPU* time. We want to emphasise that, in the figures shown shortly, we only plot results which correspond to reasonably good performances and discard those corresponding to poor performances. (Thus, for example, results corresponding to the anomolous behaviours discussed above for some of the codes are not included.) Therefore, all the figures show “almost optimal” results by all the packages. We now present the numerical results by all codes.

In this section we will consider only Problem 1, 2, 4, 6, and 7. Problem 3 will be considered in Section 5.5 to illustrate the capability of BACOL for handling problems with blow-up solutions. Problem 5 will be discussed in Section 5.4 in our investigation of the effect of differentiation of the boundary conditions.

5.3.1 Problem 1

Figures 5.13 and 5.14 show the performance of all the codes for Problem 1 with $\epsilon = 10^{-3}$ and $\epsilon = 10^{-4}$ respectively. The L^2 -norm error is computed at $T_{out} = 1$. Here

CPU means the *CPU* time in seconds and error means the L^2 -norm of the difference between the approximate solution and the exact solution.

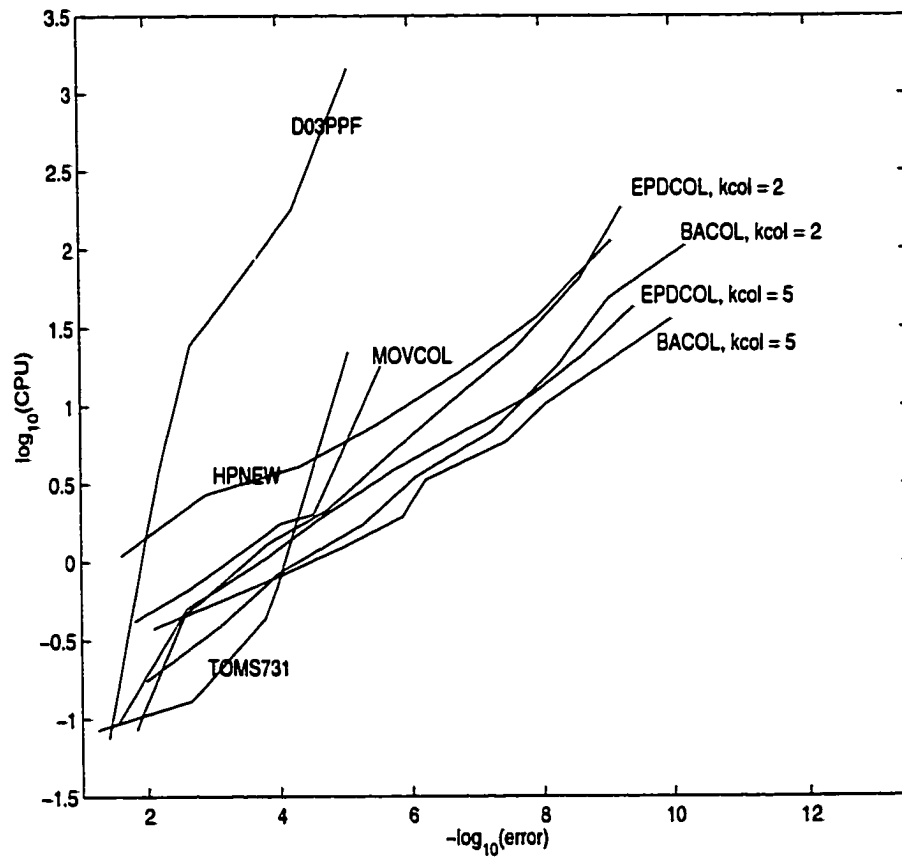


Figure 5.13: The *CPU* time and the error for Problem 1 with $\epsilon = 10^{-3}$.

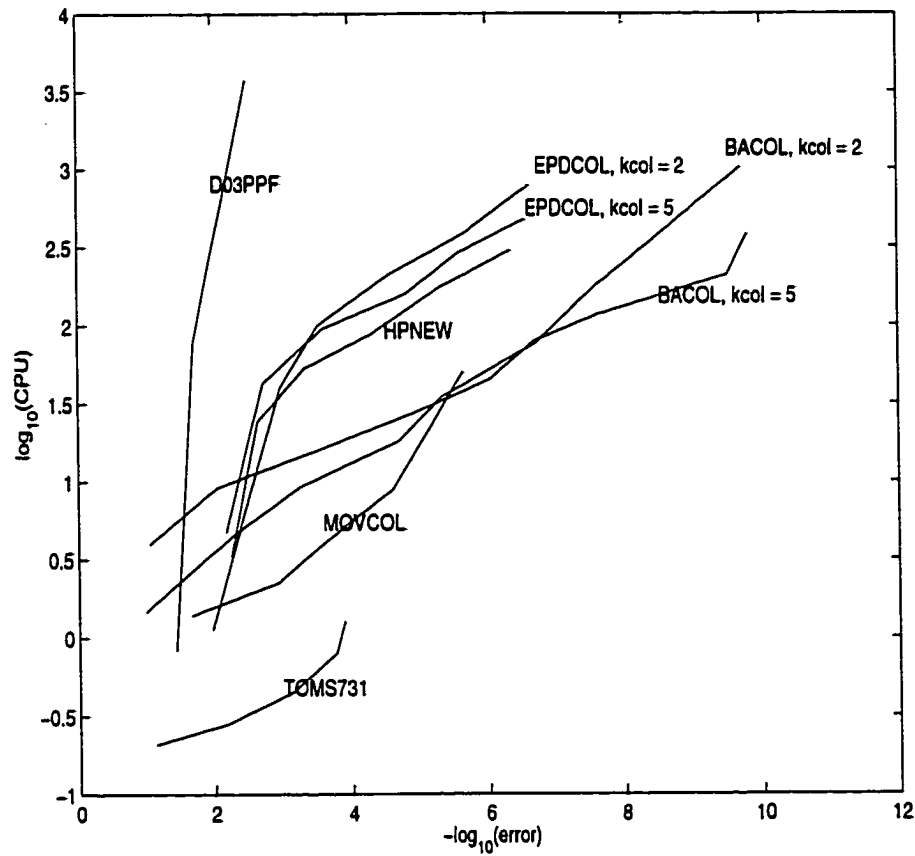


Figure 5.14: The *CPU* time and the error for Problem 1 with $\epsilon = 10^{-4}$.

We make the following observations.

- BACOL is comparable at coarse tolerances, and it is the most efficient for computing high accuracy solutions.
- When the spatial difficulty increases (i.e., ϵ changes from 10^{-3} to 10^{-4}), the fixed-mesh package, EPDCOL, compared with the adaptive packages, becomes relatively slower.
- D03PPF is the slowest software package for a fixed accuracy, and even slower than the fixed-mesh code, EPDCOL.

- When TOL is relatively coarse, TOMS731 is the most efficient (with the condition that an optimal number of mesh points is found a priori), but it is unable to produce a high accuracy solution. With $\epsilon = 10^{-3}$, the best L^2 -norm error, (10^{-5}), is obtained approximate by TOMS731; while for with $\epsilon = 10^{-4}$, this code is unable to provide an approximate solution with the error smaller than 10^{-4} .
- MOVCOL is comparable for coarse or moderate tolerances. But when we require an accuracy smaller than 10^{-6} , MOVCOL becomes less efficient. For example, consider $\epsilon = 10^{-3}$. When we choose $NINT = 150$ and $TOL = 10^{-6}$, we obtain an L^2 -norm error of 2.81×10^{-6} with $CPU = 18.21$ seconds. However, when we choose $NINT = 350$ and $TOL = 10^{-7}$, we obtain an error of 5.11×10^{-7} with $CPU = 4710.64$ seconds. This performance can be also observed in the other problems except for Problem 7 which is a relatively easy problem in term of the spatial difficulty. The reason may be that, for problems with a lot of spatial difficulties, MOVCOL is not able to compute a high accuracy approximate solution with reasonable efficiency. Therefore, we did not plot any result using MOVCOL with an accuracy request of less than 10^{-6} except for Problem 7.
- HPNEW is slow for coarse tolerances. It may be better for high tolerances, e.g., with $\epsilon = 10^{-3}$, HPNEW is faster than MOVCOL when the error is close to 10^{-5} . During our experiments, we found that for coarse tolerances HPNEW has a tendency to use many more mesh points. For example, for Problem 1 with $\epsilon = 10^{-3}$, we choose $TOL = 10^{-2}$ and HPNEW ends up with $NINT = 250$, compared with $NINT = 13$ from BACOL with $KCOL = 2$. This may due to the local hp-refinement strategy employed in HPNEW.

5.3.2 Problem 2

Figures 5.15 and 5.16 show the performance of all the codes for Problem 2 with $\epsilon = 10^{-3}$ and $\epsilon = 10^{-4}$ respectively. The L^2 -norm error is computed at $T_{out} = 1$.

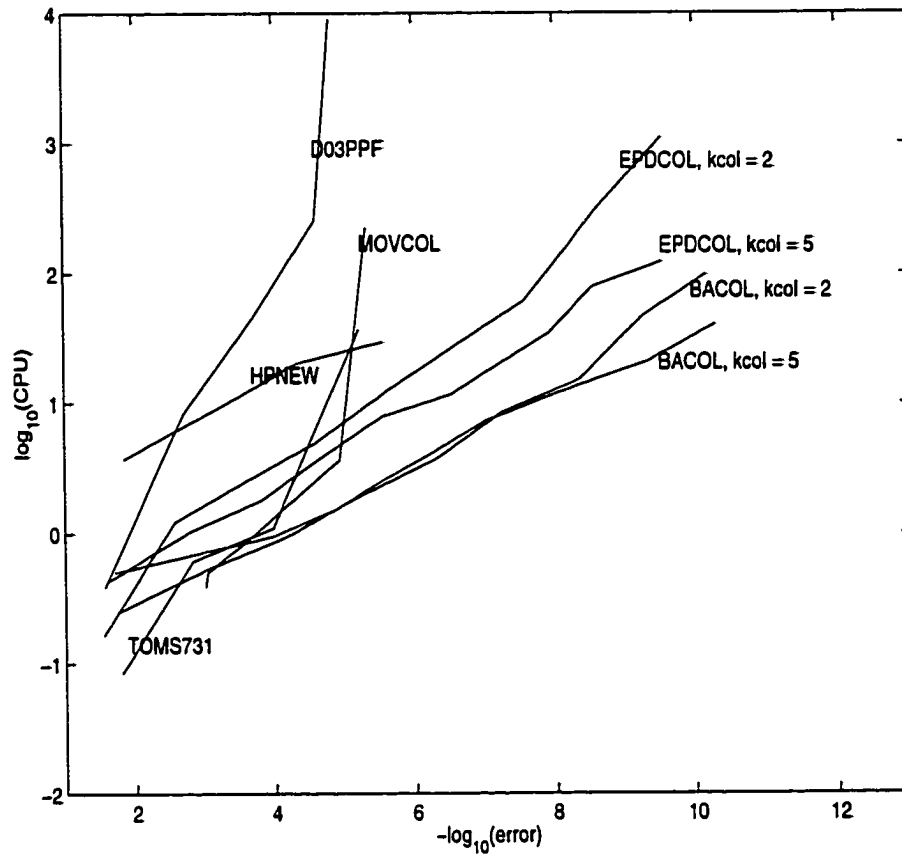


Figure 5.15: The *CPU* time and the error for Problem 2 with $\epsilon = 10^{-3}$.

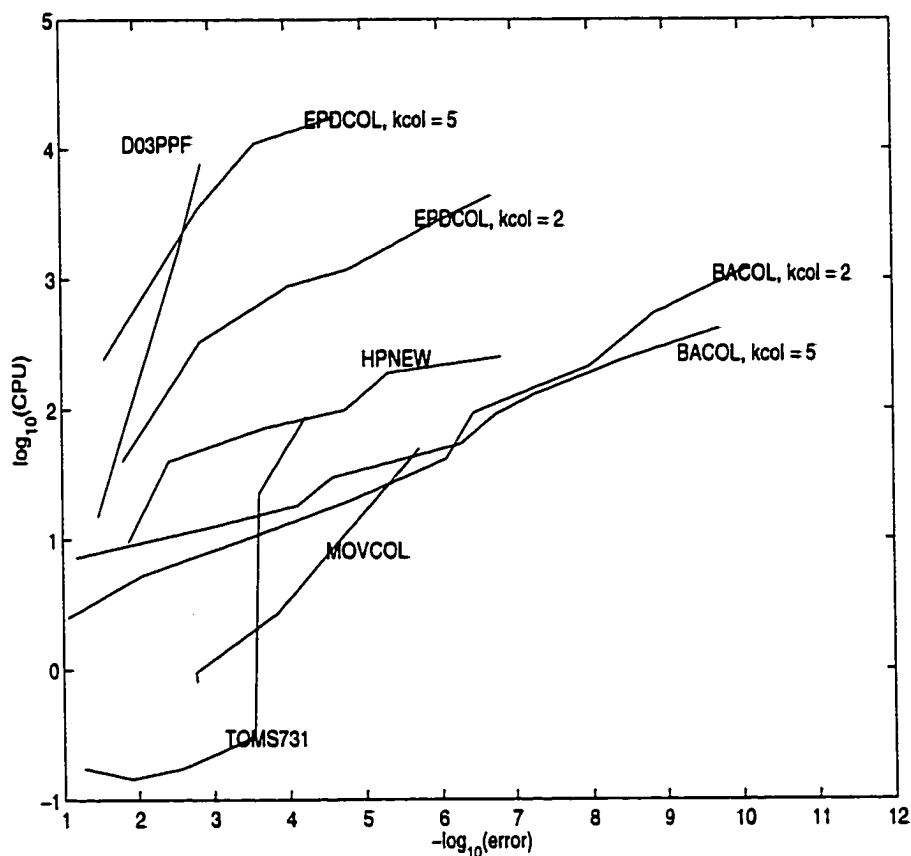


Figure 5.16: The *CPU* time and the error for Problem 2 with $\epsilon = 10^{-4}$.

We can make similar observations from Figures 5.15 and 5.16. We need to pay particular attention to Figure 5.16. In Figures 5.13-5.15, we see that EPDCOL with $KCOL = 2$ performs better using a coarse tolerance than EPDCOL with $KCOL = 5$; the latter is more efficient for the purpose of obtaining a high accuracy solution. However, things are different in Figure 5.16. As we see, EPDCOL with $KCOL = 2$ is more efficient for all the tolerances than with $KCOL = 5$. In fact, when the solution of the PDEs is *smooth* and one is using a *fixed* mesh package, it is more efficient to apply a high order discretization to the spatial domain (some numerical results can be found in [11, 52]). For Problem 2 with $\epsilon = 10^{-4}$, the solution may be considered nonsmooth. That may explain why EPDCOL with $KCOL = 5$ performs poorly for

small tolerances. However, we see that BACOL, which uses an adaptive approach, still gives better results for the high order methods when a small tolerance is supplied.

5.3.3 Problem 4

Figure 5.17 shows the performance of all the codes for Problem 4. The L^2 -norm error is computed at $T_{out} = 36$.

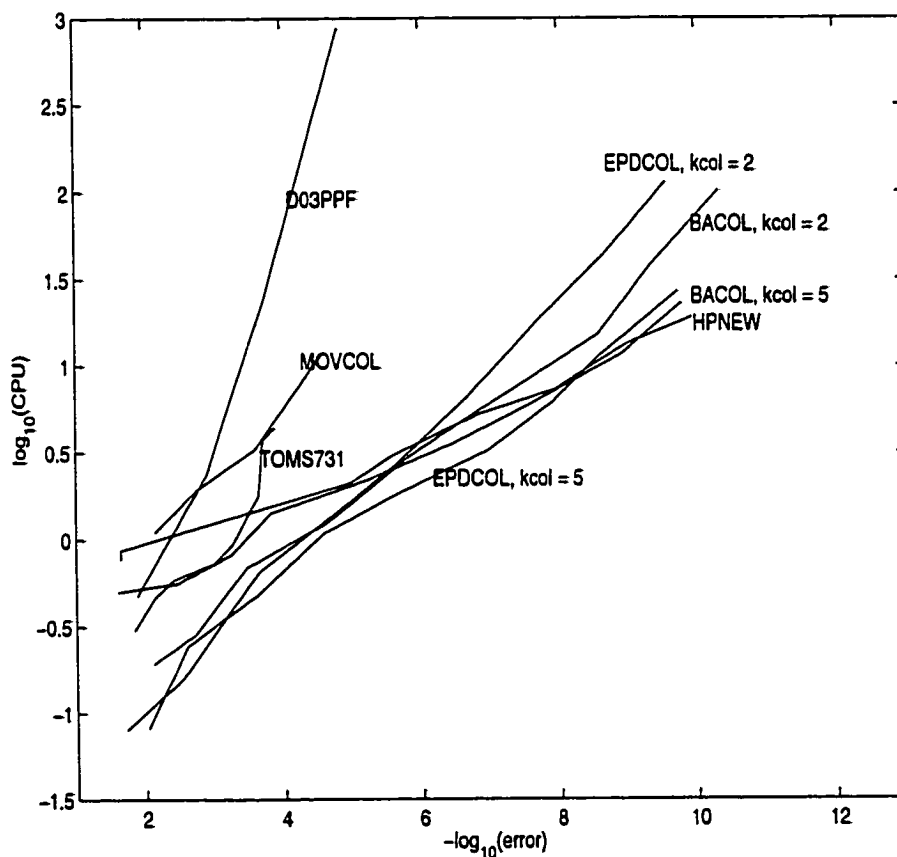


Figure 5.17: The *CPU* time and the error for Problem 4.

This is a relatively easy problem in terms of the spatial difficulty. We note that EPDCOL works very well while BACOL is comparable with EPDCOL. MOVCOL struggles with this problem, and TOMS731 is less efficient compared with EPDCOL

and BACOL. HPNEW, with a small tolerance, is the most efficient although it needs more *CPU* time when a larger tolerance is required. As usual, D03PPF is the slowest code.

5.3.4 Problem 6

We note that some of the boundary conditions (BCs) for Problem 6 are mixed, i.e., they involve conditions on both solution and first derivative components. As mentioned before, HPNEW can only handle (pure) Dirichlet or Neumann BCs. Therefore, Figure 5.18 shows the performance of all the codes except HPNEW for this problem. The L^2 -norm error is computed at $T_{out} = 18$. We first calculate the L^2 -norm error for each PDE component. The maximum L^2 -norm error is then plotted in Figure 5.18.

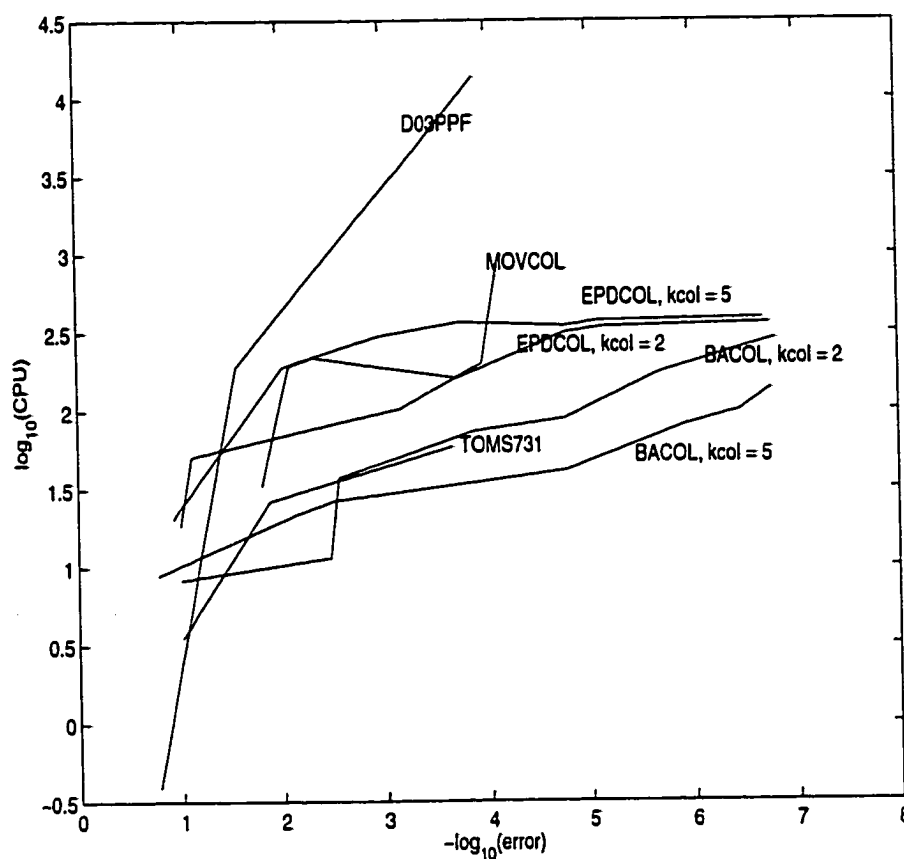


Figure 5.18: The *CPU* time and the error for Problem 6.

This problem has a smooth solution for most of the temporal domain. However, there is a wavefront moving from left to right between $t = 9$ and $t = 10$. This solution behavior should favour BACOL since it has the ability to adjust the number of mesh points while the other codes, except HPNEW, use a fixed number of mesh points throughout the computation. As expected, Figure 5.18 shows that BACOL is again the most efficient, and is comparable with TOMS731 even when a small tolerance is used.

5.3.5 Problem 7

In our experiments of Problem 7, we found that when we attempted to solve this problem using EPDCOL, it repeatedly failed the convergence test of the Newton iteration before it reached the output time $T_{out} = 0.5$; and when we attempted to solve this problem using HPNEW, it gave an error message that, after a remeshing, two adjacent mesh points of the new mesh were so close as to appear in the same place. Therefore, we show in Figure 5.19 performance results for all codes except EPDCOL and HPNEW. The L^2 -norm error is computed at $T_{out} = 0.5$.

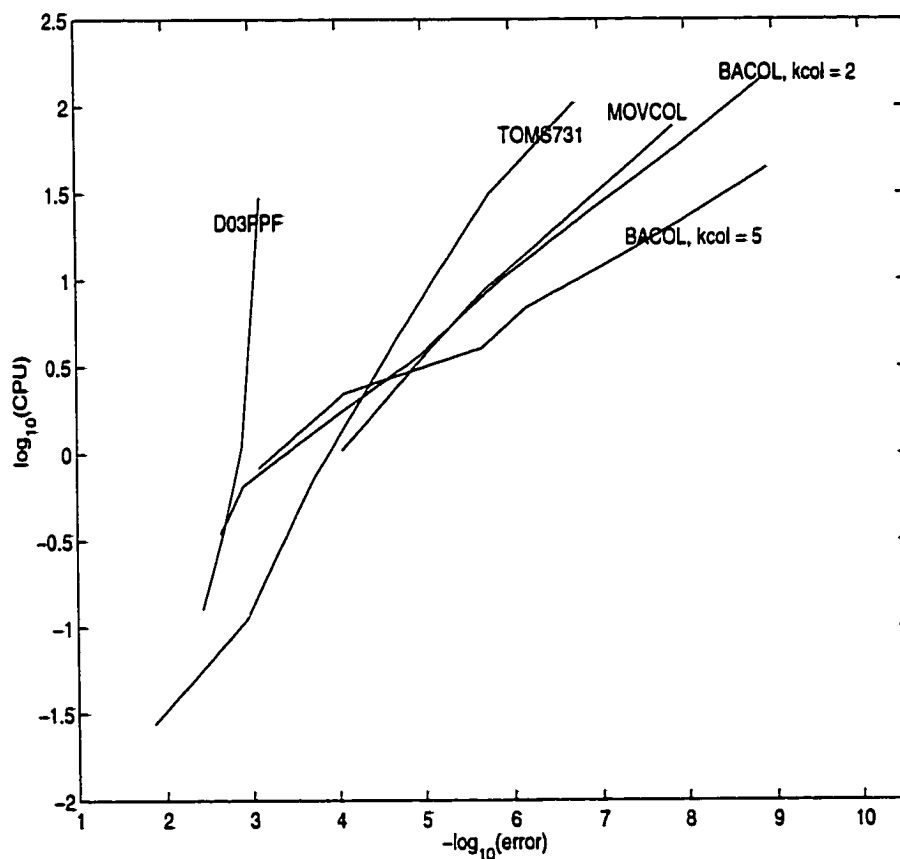


Figure 5.19: The *CPU* time and the error for Problem 7.

This is a relatively easy problem in terms of the spatial difficulty. We note that

MOVCOL is able to compute an approximate solution of the L^2 -norm error within 10^{-8} in a reasonable CPU time, which it is unable to obtain for other problems. Once again TOMS731 is more efficient when a coarse tolerance is used, and D03PPF is the slowest in all the codes. As usual, BACOL is comparable for a coarse tolerance and is more efficient when a small tolerance is used.

5.4 Inconsistent Initial and Boundary Conditions

Problem 5 is an interesting problem because the initial condition is not consistent with the boundary conditions. As mentioned before, the initial condition given for $v(t, 0)$ implies that $v_x(0, 0) = 5 \left(2 - \frac{1}{\cosh^2(20)} \right) \approx 10$, which is inconsistent with the boundary condition $v_x(0, 0) = 0$.

In [53], the authors stated that EPDCOL could not handle problems with inconsistency between the initial condition and the boundary conditions. We now plot the approximate solutions of $v(x, t)$ at $T_{out} = 35.83$ in Figure 5.20, using both BACOL and EPDCOL. We consider $TOL = 10^{-6}$ for both codes. We choose $KCOL = 2$, $NINT = 10$ for BACOL and $KCOL = 2$, $NINT = 1000$ for EPDCOL.

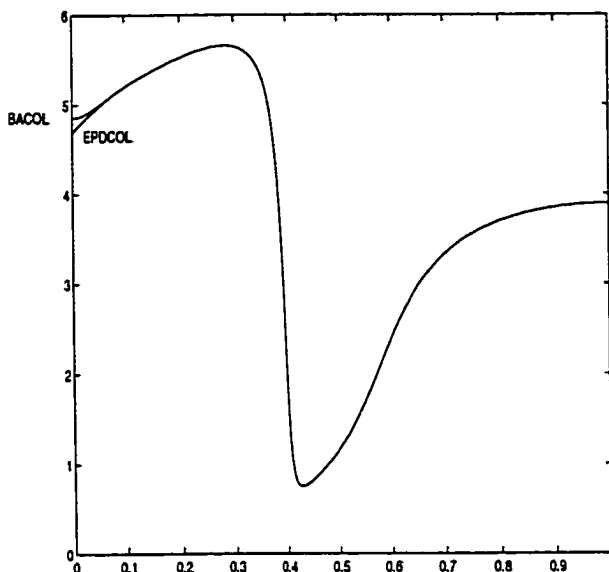


Figure 5.20: The approximate solutions using BACOL and EPDCOL.

Clearly the approximate solutions are different at the left end point. In fact we found that at $x = 0$, $v_x(0, T_{out}) = 10$ when using EPDCOL while BACOL gives $v_x(0, T_{out}) = 0$. EPDCOL gives an incorrect solution because it differentiates the BCs. Therefore, for Problem 5, EPDCOL solves the problem using

$$(v_x)_t(0, t) = 0 \quad (5.10)$$

instead of

$$v_x(0, t) = 0. \quad (5.11)$$

Moreover, EPDCOL assumes that the initial condition is consistent with the BCs. Therefore, EPDCOL uses the initial condition $v_x(0, 0) = 10$. From (5.10), we see that EPDCOL thus requires $v_x(0, t)$ to be a constant throughout the computation, i.e.,

$$v_x(0, t) = 10,$$

which leads to an incorrect approximate solution. This explains why EPDCOL is unable to handle PDEs with the inconsistency between the initial conditions and the BCs.

Furthermore, in our experiments, we found that when we attempted to solve this problem using MOVCOL, it was unable to get started because DASSL, the underlying time integrator within MOVCOL, was unable to calculate consistent derivatives for the initial step. Therefore, we show in Figure 5.21 performance results for all codes except EPDCOL and MOVCOL. The L^2 -norm error is computed at $T_{out} = 35.83$. We first calculate the L^2 -norm error for each PDE component. The maximum L^2 -norm error is then plotted in Figure 5.21.

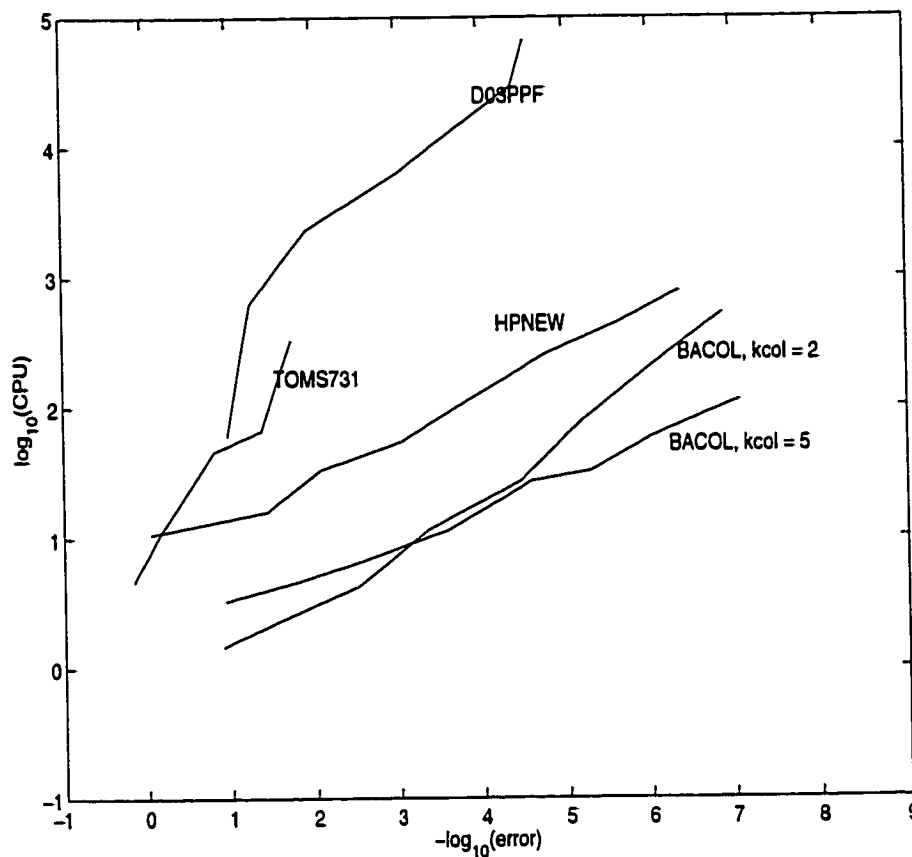


Figure 5.21: The *CPU* time and the error for Problem 5.

We observe that TOMS731 is unable to obtain an approximate solution which has an L^2 -norm error smaller than 10^{-2} . D03PPF is very slow as usual and HPNEW is about 5 times slower than BACOL.

5.5 BACOL for Problems with Blow-Up Solutions

Many physical problems have the property that as t increases, the solution goes to infinity; i.e., there exists a blow-up solution. It is a challenge for numerical methods to correctly model the blow-up behaviour. When a singularity occurs, if a fixed mesh is used to reproduce such behaviour, large number of mesh points must be

used throughout the computation in order to give a reasonable accuracy. Budd et al. [26] employed moving mesh methods to solve such problems. In this section we will illustrate that BACOL is able to handle problems with blow-up solutions.

The problem we consider here is Problem 3 with $p = 2$, introduced in Section 5.1. Using BACOL with $ATOL = RTOL = 10^{-7}$, $KCOL = 5$, and $NINT = 5$ as the initial inputs, we found that the blow-up time was approximately 0.082437272703 (which agrees with [25]), when the maximum solution was around 5.21×10^{12} . Since the solution has its maximum value at $x = 0.5$, we divide the solution by $U(0.5, t)$, which makes the maximum value 1 all times. In Figure 5.22, we plot $U(x, t)/U(0.5, t)$ at $t = 0, 0.08, 0.0824, 0.082437$. Table 5.7 shows the values of $U(0.5, t)$ at different times which clearly demonstrate the blow-up behaviour of the solution.

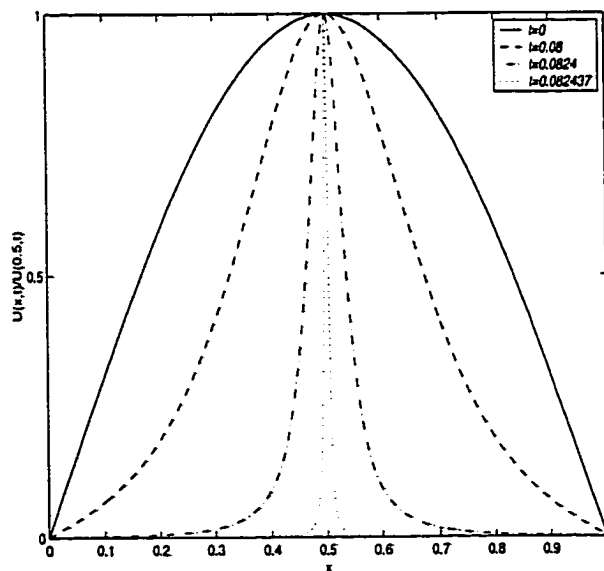


Figure 5.22: $U(x, t)/U(0.5, t)$ at $t = 0, 0.08, 0.0824, 0.082437$.

t	U(0.5,t)
0	$2.00 \cdot 10$
0.08	$4.53 \cdot 10^2$
0.0824	$2.80 \cdot 10^4$
0.082437	$3.76 \cdot 10^6$
0.08243727	$3.76 \cdot 10^8$
0.0824372727	$3.17 \cdot 10^{11}$
0.082437272703	$5.21 \cdot 10^{12}$

Table 5.7: The maximum value of the solution as t approaches the blow-up time

5.6 Numerical Investigation of Convergence Rates

De Boor and Swartz [23] studied boundary value ODEs of the form

$$y^{(m)} = f(t, y, y', \dots, y^{(m-1)}).$$

They used collocation with a B-spline basis and an approximate solution, Y , from a piecewise polynomial subspace of degree p . They then proved that, if the exact solution is sufficiently smooth so that it has $2(p+1) - m$ continuous derivatives and $p+1 - m$ Gauss-Legendre points are chosen as the collocation points in each subinterval, the following error results hold. Let h be the maximum subinterval length; then, at mesh points,

$$|Y - y| = O(h^{2(p+1-m)}), \quad (5.12)$$

while elsewhere,

$$|Y - y| = O(h^{p+1}). \quad (5.13)$$

We note that, if $p+1 > 2m$, then a higher order of convergence is obtained at the mesh points compared with the order at non-mesh points; this is referred to as *superconvergence* at the mesh points. Although, to our knowledge, there is no similar result for PDE cases, we now show some numerical results which show agreement with these rates of convergence.

In the PDE system we consider, (1.1), the highest spatial derivative is second order. This implies $m = 2$. Recall that the number of collocation points in each subinterval, $KCOL$, is equal to $p - 1$. By writing (5.12) and (5.13) in terms of $KCOL$, we obtain the similar formulas:

$$|U - u| = O(h^{2KCOL}) \quad \text{at mesh points;} \quad (5.14)$$

$$|U - u| = O(h^{KCOL+2}) \quad \text{elsewhere,} \quad (5.15)$$

where u is the exact solution for the PDEs in (1.1), and U is the approximate solution. We now show some numerical results which indicate that there does exist superconvergence at the mesh points, and the expected, lower rate of convergence at other points.

We now consider a simple problem, the second example in [53]:

$$u_t = u_{xx} + \pi^2 \sin \pi x, \quad 0 < x < 1, \quad t > 0, \quad (5.16)$$

$$u(x, 0) = 1, \quad 0 \leq x \leq 1,$$

$$u(0, t) = u(1, t) = 1, \quad t > 0.$$

The exact solution for this problem is easily seen to be

$$u(x, t) = 1 + \sin \pi x (1 - e^{-\pi^2 t}).$$

So as not to use BACOL, we will employ EPDCOL for this numerical experiment. We recall that EPDCOL employs the same spatial discretization as BACOL. We will now apply EPDCOL on the above problem with uniform mesh and $KCOL$ Gauss-Legendre collocation points in each subinterval. If we choose $NINT = N$ and $T_{out} = 1$, the error at the i -th internal mesh point, $x_i = i/N$, $i = 1, \dots, N - 1$ is then calculated by $E_i = |U(x_i, T_{out}) - u(x_i, T_{out})|$. Since the above problem has the constant Dirichlet boundary conditions at both boundaries, the error at the boundary points will be around machine epsilon. Therefore, we will not calculate the error at the

boundary points. We will also consider $NINT = 2N$ and $NINT = 4N$, and we can calculate the corresponding errors at $x_i = i/N$, $i = 1, \dots, N-1$, which we will refer to as $\acute{E}(i)$ and $\grave{E}(i)$, respectively. We note that the original x_i corresponding to $NINT = N$, are also mesh points for $NINT = 2N$ or $4N$. Now, if (5.14) holds, we should have

$$\frac{|E_i - \acute{E}(i)|}{|\acute{E}(i) - \grave{E}(i)|} \approx \frac{h^2 KCOL - (\frac{h}{2})^2 KCOL}{(\frac{h}{2})^2 KCOL - (\frac{h}{4})^2 KCOL} = 2^2 KCOL, \quad (5.17)$$

where $i = 1, \dots, N-1$. Obviously from (5.17) we have

$$C_1 = \max_{1 \leq i \leq N-1} \frac{|E_i - \acute{E}(i)|}{|\acute{E}(i) - \grave{E}(i)|} \approx 2^2 KCOL. \quad (5.18)$$

For confirmation of superconvergence, we still need to show the rate of convergence at non-mesh points. Instead of calculating the error in terms of the maximum norm, we choose to calculate the L^2 -norm error between the exact solution $u(x, t)$ and the approximate solution $U(x, t)$, using

$$\int_0^1 (U(x, t) - u(x, t))^2 dx = \sum_{i=1}^{NINT} \int_{x_i}^{x_{i+1}} (U(x, t) - u(x, t))^2 dx, \quad (5.19)$$

and

$$\int_{x_i}^{x_{i+1}} (U(x, t) - u(x, t))^2 dx \approx (x_{i+1} - x_i) \sum_{r=1}^{KCOL+3} w_r (U(v_r, t) - u(v_r, t))^2, \quad (5.20)$$

for $i = 0, \dots, NINT$. The $\{w_r\}_{r=1}^{KCOL+3}$ are the Gauss-Legendre quadrature weights on $[0, 1]$, and $v_r = x_i + (x_{i+1} - x_i)\rho_r$, where $\{\rho_r\}_{r=1}^{KCOL+3}$ are the Gauss-Legendre quadrature points on $[0, 1]$.

From (5.15), we expect to obtain

$$\int_0^1 (U(x, t) - u(x, t))^2 dx = O(h^{KCOL+2}). \quad (5.21)$$

We compute the L^2 -norm errors, namely, $\|E\|_2$, $\|\acute{E}\|_2$, $\|\grave{E}\|_2$ for $NINT = N, 2N, 4N$. If (5.21) holds, a similar calculation gives,

$$C_2 = \frac{\|E\|_2 - \|\dot{E}\|_2}{\|\dot{E}\|_2 - \|\ddot{E}\|_2} \approx 2^{KCOL+2}. \quad (5.22)$$

Table 5.8 shows the values of C_1 and C_2 for (5.16) for different $KCOL$ and N values. We denote $RTOL$ for the relative tolerance, and recall that EPDCOL only requires the relative tolerance and that this tolerance is applied only to the time integration. $RTOL$ is chosen such that it is much smaller than the L^2 -norm error and the error at the mesh points. Therefore, the error which comes from the time integration can be ignored.

$KCOL$	N	$RTOL$	C_1	C_2
2	4	10^{-8}	16.43	16.29
2	5	10^{-8}	16.27	16.19
2	6	10^{-9}	16.20	16.13
3	4	10^{-10}	63.47	32.13
3	5	10^{-11}	64.71	32.08
3	6	10^{-11}	64.99	32.06
4	2	10^{-15}	279.4	63.66
4	3	10^{-15}	267.7	63.84
4	4	10^{-15}	276.2	63.91

Table 5.8: Confirmation of superconvergence.

As expected, the values of C_1 and C_2 agree with (5.18), (5.22) very well.

Chapter 6

Conclusions and Future Work

In this thesis, BACOL, an adaptive MOL software package for solving one-dimensional (1-D) parabolic PDEs is presented. Numerical experiments are done using BACOL and several other adaptive or non-adaptive software packages on a number of test problems. The results presented in Chapter 5 illustrate that when a high accuracy is required, BACOL is clearly the most efficient among all the packages; and for a coarse tolerance, BACOL is still one of the fastest codes for most test problems. We note that except for BACOL and HPNEW, all the other codes do not have the ability to change the number of mesh points and thus control the spatial error. Therefore, in general, for codes except BACOL and HPNEW, it is difficult to achieve the optimal performances shown in Chapter 5. Comparing BACOL and HPNEW, BACOL is clearly more efficient than HPNEW for most test problems. In addition to efficiency issues, we have observed various difficulties using other codes (for example, EPDCOL and MOVCOL are unable to solve problems with inconsistencies between the initial condition and the BCs; while HPNEW is unable to solve problems with mixed BCs), compared to BACOL.

There is much left for future research. A summary of future work follows.

- It should be possible to develop an algorithm to vary the degrees of the piecewise polynomials employed in the spatial discretization. As shown in Chapter 5, for

a given problem and a given tolerance, it takes different amounts of *CPU* time when we apply BACOL with different *KCOL* values. In general, the smaller the required tolerance, the more efficient it is to use BACOL with a larger *KCOL* value. It is easy to see that the number of multiplications in each time step is linear with respect to the number of subintervals; i.e., $C_{KCOL} * NINT_{KCOL}$, where C_{KCOL} is a constant depending on *KCOL*, and $NINT_{KCOL}$ is the number of subintervals which is necessary to achieve a given *TOL* using *KCOL*. On the other hand, we have the relationship, approximately, $(NINT_{KCOL})^{-(KCOL+2)} = \text{constant}$, which is based on (5.8). Therefore, if we can estimate C_{KCOL} , we may be able to vary the degrees of the piecewise polynomials depending on the value of $NINT_{KCOL}$.

- It would be interesting to attempt to extend BACOL to more general 1-D cases, e.g., a parabolic system coupled with elliptic equations or ordinary differential equations. This may involve some changes in the interface; e.g., there will not be enough initial conditions or boundary conditions. For instance, for a parabolic system coupled with elliptic equations, some variables may only have boundary conditions but do not have initial conditions. This will not change the structure required by COLROW; however, the process to obtain the initial conditions for DASSL will be different, and the scaling technique would be applied to the algebraic constraints which are generated from the elliptic equations.
- It will be interesting to see whether a similar strategy will work for higher dimensional cases or not. Unfortunately collocation on complex regions is difficult because much of the structure of the linear system is lost. It may be better to use a Galerkin method that involves less continuity. In case of rectangle domain and quasilinear PDEs, it may be possible to apply collocation using a tensor product basis (see [71]).

Bibliography

- [1] S. Adjerid, M. Aiffa, and J.E. Flaherty. High-order finite element methods for singularly-perturbed elliptic and parabolic problems. *SIAM J. Appl. Math.*, 55:520–543, 1995.
- [2] S. Adjerid and J.E. Flaherty. A moving finite element method with error estimation and refinement for one-dimensional time dependent partial differential equations. *SIAM J. Numer. Anal.*, 23:778–796, 1986.
- [3] S. Adjerid and J.E. Flaherty. A moving-mesh finite element method with local refinement for parabolic partial differential equations. *Comput. Meth. Appl. Mech. Engrg.*, 55:3–26, 1986.
- [4] S. Adjerid, J.E. Flaherty, P.K. Moore, and Y. Wang. High-order adaptive methods for parabolic systems. *Physica D*, 60:94–111, 1992.
- [5] S. Adjerid, J.E. Flaherty, and Y. Wang. A posteriori error estimation with finite element methods of lines for one-dimensional parabolic systems. *Numer. Math.*, 65:1–21, 1993.
- [6] U. Ascher, J. Christiansen, and R.D. Russell. Collocation software for boundary value ODEs. *ACM Trans. Math. Softw.*, 7:209–222, 1981.
- [7] U. Ascher, R.M.M Mattheij, and R.D. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. SIAM, Philadelphia, 1995.
- [8] U. Ascher, S. Pruess, and R.D. Russell. On spline basis selection for solving differential equations. *SIAM J. Numer. Anal.*, 20:121–141, 1983.
- [9] U. Ascher and R.D. Russell. Reformulation of boundary value problems into "standard" form. *SIAM Review*, 23:238–254, 1981.
- [10] M. Berzins, P.J. Capon, and P.K. Jimack. On spatial adaptivity and interpolation when using the method of lines. *Appl. Num. Math.*, 26:117–133, 1998.

- [11] M. Berzins and P.M. Dew. Algorithm 690: Chebyshev polynomial software for elliptic-parabolic systems of PDEs. *ACM Trans. Math. Softw.*, 17:178–206, 1991.
- [12] M. Berzins, P.M. Dew, and R.M. Furzeland. Software for time-dependent problems. In B. Engquist and T. Smedsaas, editors, *PDE Software: Modules, Interfaces and Systems*, pages 309–324; Amsterdam, 1984. IFIP/North-Holland.
- [13] M. Berzins, P.M. Dew, and R.M. Furzeland. Developing software for time-dependent problems using the method of lines and differential-algebraic integrators. *Appl. Num. Math.*, 5:375–397, 1989.
- [14] M. Berzins and R.M. Furzeland. A type-insensitive method for the solution of stiff and non-stiff differential equations. Technical report, Department of Computer studies, Leeds Univeristy, England, 1986.
- [15] M. Berzins and R.M. Furzeland. An adaptive theta method for the solution of stiff and non-stiff differential equations. *Appl. Num. Math.*, 9:1–19, 1992.
- [16] M. Bieterman and I. Babuška. The finite element method for parabolic equations. I. a posteriori error estimation. *Numer. Math.*, 40:339–371, 1982.
- [17] M. Bieterman and I. Babuška. The finite element method for parabolic equations. II. a posteriori error estimation and adaptive approach. *Numer. Math.*, 40:373–406, 1982.
- [18] J.G. Blom, J.M. Sanz-Serna, and J.G. Verwer. On simple moving grid method for one-dimensional evolutionary partial differential equations. *J. Comput. Phys.*, 74:191–213, 1988.
- [19] J.G. Blom and P.A. Zegeling. Algorithm 731: A moving-grid interface for systems of one-dimensional time-dependent partial differential equations. *ACM Trans. Math. Softw.*, 20:194–214, 1994.
- [20] R.F. Boisvert, S.E. Howe, and D.K. Kahaner. GAMS: A framework for the management of scientific software. *ACM Trans. Math. Softw.*, 11:313–355, 1985.
- [21] C.De Boor. Package for calculating with B-Splines. *SIAM J. Numer. Anal.*, 14:441–472, 1977.
- [22] C.De Boor. *A Pratical Guide to Splines*. Springer-Verlag, New York, 1978.
- [23] C.De Boor and B. Swartz. Collocation at Gaussian points. *SIAM J. Numer. Anal.*, 10:582–606, 1973.

- [24] K.E. Brenan, S.L. Campbell, and L.R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. SIAM, Philadelphia, 1996.
- [25] C.J. Budd, J. Chen, W. Huang, and R.D. Russell. Moving mesh methods with applications to blow-up problems for PDEs. In D.F. Griffiths and G.A. Watson, editors, *Numerical Analysis 1995: Proc. of 1995 Biennial Conference on Numerical Analysis*, pages 1–17, 1996.
- [26] C.J. Budd, W. Huang, and R.D. Russell. Moving mesh methods for problems with blow-up. *SIAM J. Sci. Comput.*, 17:305–327, 1996.
- [27] G.D. Byrne and A.C. Hindmarsh. A polyalgorithm for the numerical solution of ordinary differential equations. *ACM Trans. Math. Softw.*, 1:71–96, 1975.
- [28] J. Carroll. A composite integration scheme for the numerical solution of systems of parabolic pdes in one space dimension. *Journal of Computational and Applied Mathematics*, 46:327–343, 1993.
- [29] M.B. Carver. The choice of algorithms in automated method of lines solution of partial differential equations. In L. Lapidus and W.E. Schiesser, editors, *Numerical Methods for Differential Systems*, pages 243–265. Academic Press, Inc., 1976.
- [30] 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.*, 9:358–375, 1983.
- [31] G. Fairweather. *Finite Element Galerkin Methods for Differential Equations*. Marcel Dekker, Inc., New York, 1973.
- [32] W. Gui and I. Babuška. The h, p and h-p versions of the finite element method in 1 dimension. Part I. the error analysis of the p-version. *Numer. Math.*, 49:577–612, 1986.
- [33] W. Gui and I. Babuška. The h, p and h-p versions of the finite element method in 1 dimension. Part II. the error analysis of the h- and h-p versions. *Numer. Math.*, 49:613–657, 1986.
- [34] W. Gui and I. Babuška. The h, p and h-p versions of the finite element method in 1 dimension. Part III. the adaptive h-p version. *Numer. Math.*, 49:659–683, 1986.
- [35] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer-Verlag, Berlin, 1991.

- [36] A.C. Hindmarsh. Preliminary documentation of GEARIB: Solution of implicit systems of ordinary differential equations with banded Jacobian. Technical Report UCID-30130, Lawrence Livermore National Laboratory, 1976.
- [37] A.C. Hindmarsh. GEARB: Solution of ordinary differential equations having banded jacobian. Technical Report UCID-30059, Lawrence Livermore National Laboratory, 1977.
- [38] A.C. Hindmarsh. ODEPACK: A systematized collection of ODE solvers. In R. Stepleman et al., editor, *Scientific Computations*, pages 55–64. IMACS/North-Holland Publishing Company, 1983.
- [39] W. Huang, Y. Ren, and R.D. Russell. Moving mesh methods based on moving mesh partial differential equations. *J. Comput. Phys.*, 113:279–290, 1994.
- [40] W. Huang, Y. Ren, and R.D. Russell. Moving mesh partial differential equations (MMPDEs) based on the equidistribution principle. *SIAM J. Numer. Anal.*, 31:709–730, 1994.
- [41] W. Huang and R.D. Russell. A moving collocation method for solving time dependent partial differential equations. *Appl. Numer. Math.*, 20:101–116, 1996.
- [42] W. Huang and R.D. Russell. Analysis of moving mesh partial differential equations with spatial smoothing. *SIAM J. Numer. Anal.*, 34:1106–1126, 1997.
- [43] M. Humi and W.B. Miller. *Boundary Value Problems and Partial Differential Equations*. PWS-KENT, Boston, 1992.
- [44] M. Hyman. The method of lines solution of partial differential equations. Technical report, Courant Inst. Mathematics Sciences, 1976.
- [45] P. Keast, 1992. Private communication.
- [46] P. Keast and P.H. Muir. Algorithm 688. EPDCOL: A more efficient PDECOL code. *ACM Trans. Math. Softw.*, 17:153–166, 1991.
- [47] P. Keast, P.H. Muir, and T.B. Nokonechny. A method of lines package, based on monomial spline collocation, for systems of one dimensional parabolic differential equations. In *Numerical Analysis (A.R. Mitchell 75th birthday volume)*, pages 207–224. World Scientific Publishing, 1996.
- [48] J. Lawson, M. Berzins, and P.M. Dew. Balancing space and time errors in the method of lines for parabolic equations. *SIAM J. Sci. Stat. Comput.*, 12:573–594, 1991.

- [49] B. Leimkuhler, L. Petzold, and C.W. Gear. Approximation methods for the consistent initialization of differential-algebraic equations. *SIAM J. Numer. Anal.*, 28:205–226, 1991.
- [50] M. Machura and R.A. Sweet. A survey of software for partial differential equations. *ACM Trans. Math. Softw.*, 6:461–488, 1980.
- [51] N.K. Madsen and R.F. Sincovec. *PDEPACK: Partial Differential Equations Package Users Guide*. Scientific Computing Consulting Services, 1975.
- [52] N.K. Madsen and R.F. Sincovec. General software for partial differential equations. In L. Lapidus and W.E. Schiesser, editors, *Numerical Methods for Differential Systems*, pages 229–242. Academic Press, New York, 1976.
- [53] N.K. Madsen and R.F. Sincovec. Algorithm 540. PDECOL, general collocation software for partial differential equations. *ACM Trans. Math. Softw.*, 5:326–351, 1979.
- [54] K. Miller. Moving finite elements. II. *SIAM J. Numer. Anal.*, 18:1033–1057, 1981.
- [55] K. Miller and R. N. Miller. Moving finite elements. I. *SIAM J. Numer. Anal.*, 18:1019–1032, 1981.
- [56] A.R. Mitchell. *The Finite Difference Method in Partial Differential Equations*. Wiley, New York, 1980.
- [57] P.K. Moore. A posteriori error estimation with finite element semi- and fully discrete methods for nonlinear parabolic equations in one space dimension. *SIAM J. Numer. Anal.*, 31:149–169, 1994.
- [58] P.K. Moore. Comparison of adaptive methods for one dimensional parabolic systems. *Appl. Numer. Math.*, 16:471–488, 1995.
- [59] P.K. Moore. Interpolation error-based a posteriori error estimation for two-point boundary value problems and parabolic equations in one space dimension. unpublished software, <http://faculty.smu.edu/pmoore/papers.html>, 2001.
- [60] T.B. Nokonechny. The method of lines using monomial spline collocation for parabolic partial differential equations. Master's thesis, Dalhousie University, 1995.
- [61] L.R. Petzold. A description of DASSL: A differential/algebraic system solver. Technical report, Sandia Labs, Livermore, CA, 1982.

- [62] L.R. Petzold and P. Lötstedt. Numerical solution of nonlinear differential equations with algebraic constraints II: Practical implications. *SIAM J. Sci. Stat. Comput.*, 7:720–733, 1986.
- [63] A. Prothero and A. Robinson. On the accuracy and stability of one step methods for solving stiff systems of ordinary differential equations. *Math. Comp.*, 28:145–162, 1974.
- [64] J.M. Sanz-Serna and I. Christie. A simple adaptive technique for nonlinear wave problems. *J. Comput. Phys.*, 67:348–360, 1986.
- [65] W.E. Schiesser. A digital simulation system for mixed ordinary/partial differential equation modes. In *Proc. Symp. Digital Simulation of Continuous Processors*, Győr, Hungary, 1971.
- [66] W.E. Schiesser. *DSS/2-An Introduction to the Numerical Method of Lines Integration of Partial Differential Equations*. Lehigh University, Pennsylvania, 1976.
- [67] W.E. Schiesser. *The Numerical Method of Lines*. Academic Press, Inc., 1991.
- [68] C. Shu. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. Technical report, NASA Langley Research Center, 1997.
- [69] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, Berlin, 1991.
- [70] R. Wang. High order adaptive method of lines for 1-D parabolic equations. Master's thesis, Dalhousie University, 1999.
- [71] Y. Wang. *A Parallel Collocation Method for Two Dimensional Linear Parabolic Separable Partial Differential Equations*. PhD thesis, Dalhousie University, 1994.
- [72] A.B. White. On selection of equidistributing meshes for two-point boundary-value problems. *SIAM J. Numer. Anal.*, 16:472–502, 1979.