HIGH ORDER ADAPTIVE METHOD OF LINES FOR 1-D PARABOLIC EQUATIONS

By

Rong Wang

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
AT
DALHOUSIE UNIVERSITY
HALIFAX, NOVA SCOTIA
AUGUST 1999

DALHOUSIE UNIVERSITY
DEPARTMENT OF
MATHEMATICS AND STATISTICS

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "**HIGH ORDER ADAPTIVE METHOD OF LINES FOR 1-D PARABOLIC EQUATIONS**" by **Rong Wang** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: August 1999

Supervisor: _____
Patrick Keast

Readers: _____
Paul Muir

_____
Shigui Ruan

DALHOUSIE UNIVERSITY

Date: **August 1999**

Author: **Rong Wang**

Title: **HIGH ORDER ADAPTIVE METHOD OF LINES FOR 1-D PARABOLIC EQUATIONS**

Department: **Mathematics and Statistics**

Degree: **M.Sc.** Convocation: **October** Year: **1999**

I dedicate this thesis

to

Fred and Miranda.

Contents

List of Tables

List of Figures

acknowledgment

I would like to thank to my supervisor, Dr. Pat Keast, for giving me a lot of valuable comments and helping me to solve the difficulties during my programming. I am also thankful to Dr. Paul Muir and Dr. Shigui Ruan for some helpful comments and suggestions.

Abstract

In this thesis an adaptive algorithm, based on the method-of-lines package, EPDCOL, for solving one dimensional parabolic partial differential equations is implemented. An approximate solution is calculated in a piecewise polynomial subspace of degree p, and an *a posteriori* error estimation of the spatial error is obtained by using a degree $p+1$ piecewise polynomial. An equidistribution principle is employed to adapt the mesh, if necessary.

The algorithm on which this approach is based is presented in Chapter 3. After a fixed time interval, which is a user supplied parameter, an L^2 norm approximate error is computed and a remeshing is done if the error requirement is not met. If the error requirement is not met after three consecutive remeshings, the code can choose to double the number of subintervals in the spatial mesh, or to increase the degree of the piecewise polynomial approximation. Computational results indicate that the error approximation converges to the true error, and our code has proved to be reliable and much more accurate than EPDCOL in dealing with problems having solutions with rapid variation.

Chapter 1

Introduction

1.1 The Problem Class

In this thesis a system of $NPDE$ parabolic partial differential equations (PDEs) is considered, and a spatial mesh refinement algorithm is provided to augment the conventional method of lines approach. This code is experimental and is limited to $NPDE = 1$, although it is easy to generalize it to systems of PDEs. We will first present the traditional method of lines appoach.

Assume that $x \in [0, 1]$ and $t \geq 0$. We assume the problem is of the form

$$u_t(t, x) = f(t, x, u(t, x), u_x(t, x), u_{xx}(t, x)), \qquad 0 \leq x \leq 1, \quad t \geq 0, \tag{1.1}$$

where the initial condition is given by

$$u(0, x) = u_0(x), \qquad 0 \leq x \leq 1, \tag{1.2}$$

and the boundary conditions are given by

$$b_L(t, u(t, 0), u_x(t, 0)) = 0, \quad t \geq 0, \tag{1.3}$$

$$b_R(t, u(t, 1), u_x(t, 1)) = 0, \quad t \geq 0, \tag{1.4}$$

where $u_t = \partial u/\partial t$ etc. Throughout the thesis, we assume that $NPDE = 1$ and that there exists a unique solution for the problem (1.1)-(1.4).

1.2 The Method of Lines

The method of lines (MOL) involves two steps. The first step is the spatial discretization, in which the spatial variables are discretized by means of finite difference methods or finite element methods. After the spatial discretization, a system of ordinary differential equations (ODEs) is generated. The second step is to integrate this ODE system by some standard ODE solver. Generally, there are three kinds of ODE solvers, Runge-Kutta methods (see, e.g. [B87]), Backward Differentiation Formulas (BDF) methods (see, e.g. [G71]) and the general multi-step methods (see, e.g. LSODE [H83]). In some application, the system of ODEs is augmented by a set of algebraic constraints and the combined system is referred to as a set of differential algebraic equations (DAEs). There is a package, DASSL [P82], designed for DAEs. Most ODE software packages are able to select the step-size automatically and many are highly efficient and reliable.

Several MOL packages have been developed for one dimensional PDE systems like (1.1)-(1.4) (see e.g., [MS79] and a modification of this, [KM91]). The traditional MOL approach fixes the mesh points and controls only the temporal error. It is robust for problems whose solutions have no regions of rapid spatial variation, or whose solutions have regions of rapid change where the regions do not move and are known a priori. However, for problems with rapidly changing solutions, such as travelling wavefronts, a fixed grid will be quite inefficient since the spatial error will dominate the temporal error, and many more mesh points are usually required to achieve high-accuracy approximate solutions. In such cases, adaptive and moving grid methods

are usually more efficient. In the rest of Chapter 1 we will discuss some standard method of lines approaches and, especially, the B-spline collocation approach, which we will use in this thesis. Then, in Chapter 2 we will review some adaptive techniques which have been used in the last twenty years. In Chapter 3 we will give a detailed discussion of our implementation of one of these techniques coupled with a method of lines algorithm.

1.3 Spatial Discretization Schemes for the Method of Lines

As mentioned before, a fundamental part of the MOL approach is the spatial discretization. In this section a survey of several standard discretization schemes is presented. There are two approaches, finite difference methods and finite element methods.

1.3.1 Finite Difference Methods

We conside the problem (1.1)-(1.4). By using a uniform mesh with mesh size h, we have $\Pi = \{x_i\}_{i=0}^{NINT}$, where $x_i = ih$ and where $NINT = 1/h$. Suppose that $U_i(t)$ is the approximate solution for $u(x_i, t)$ for $i = 0, \cdots, NINT$. By using finite difference schemes, we are able to get the approximate for $f(t, x, u, u_x, u_{xx})$. For example, suppose that

$$f(t, x, u, u_x, u_{xx}) = \left(\frac{\partial u}{\partial x}\right)\left(\frac{\partial^2 u}{\partial x^2}\right),$$

and that $u(x,t)$ is specified at the boundaries, i.e. we have Dirichlet boundary conditions. If we use the upwinded formula

$$\frac{\partial U_i}{\partial x}(t) = \frac{U_{i+1}(t) - U_i(t)}{h}, \qquad i = 1, \cdots, NINT - 1,$$

it is easy to show that this formulation is $O(h)$. However, if the central difference formula is used, we have

$$\frac{\partial U_i}{\partial x}(t) = \frac{U_{i+1}(t) - U_{i-1}(t)}{2h}, \qquad i = 1, \cdots, NINT - 1.$$

This approximation can easily be shown to be $O(h^2)$. The second derivative can be approximated by

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

and it can also be shown to be $O(h^2)$. By looking at Taylor series expansions of $u(t,x)$, we can easily construct higher order approximations, although special approximations on the boundaries are usually needed since we must obtain the same order of approximation at the boundary points. After the spatial discretization scheme is applied, a system of initial ODEs is generated as

$$\frac{d\vec{U}}{dt}(t) = \vec{F}(t, \vec{U}), \tag{1.5}$$

where $\vec{U} = [U_1(t), \cdots, U_{NINT-1}(t)]^T$, $\vec{F}(t) = [F_1, \cdots, F_{NINT-1}]^T$. Here F_i approximates $f(t, x, u, u_x, u_{xx})$ at x_i, by substituting for all spatial derivatives with finite differences.

1.3.2 Finite Element Methods

(1) *Galerkin Methods*

Before we discuss Galerkin methods, we first give some definitions of the piecewise polynomial spaces, in which we seek an approximate solution. We first assume that

the interval $[0,1]$ is to be subdivided into $NINT$ subintervals. For $\kappa > \nu \geq 0$, we denote the mesh

$$\Pi := \{0 = x_0 < x_1 < \cdots < x_{NINT} = 1\}$$

and the approximation subspace

$$S_\kappa^\nu := \{W(x) \mid W \in C^{\nu-1} \text{ for } x \in [0,1],\ \ W \in P_\kappa \text{ for } x \in (x_i, x_{i+1}), \quad i = 0, \cdots, NINT-1\},$$

where P_κ is the space of polynomials of degree κ. It is easy to see the dimension of S_κ^ν is $N = (\kappa - \nu + 1)NINT + \nu$. If some $\{\psi_i(x)\}_{j=1}^N$ is a basis for S_κ^ν, we can approximate the solution $u(x,t)$ as $U(x,t) = \sum_{j=1}^N \psi_j(x)\phi_j(t)$.

There are several standard types of Galerkin methods, such as L^2-Galerkin [T84], H^1-Galerkin [F78, T84] and H^{-1}-Galerkin [T84, W75]. Here we only review L^2-Galerkin. Suppose that we consider the problem, given by the quasilinear parabolic PDE

$$u_t = (a(x)u_x)_x + f(t,x,u), \qquad 0 \leq x \leq 1, \qquad t > 0,$$

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

$$u(0,t) = u(1,t) = 0, \qquad t > 0,$$

and suppose that there exist positive constants c_0, c_1 such that, for all $0 \leq x \leq 1$,

$$0 < c_0 \leq a(x) \leq c_1.$$

When using L^2-Galerkin, we try to find an approximate solution $U(t,x)$, which satisfies the weak form of the PDE

$$(U_t, V) = ((aU_x)_x, V) + (f, V), \qquad 0 \leq x \leq 1, \qquad t > 0, \tag{1.6}$$

where $U(x,0)$ is chosen so that $U(x,0) - u_0(x)$ is small and $(f,g) = \int_0^1 f(x)g(x)dx$. Then, integrating by parts, we obtain

$$(U_t, V) + (a(x)U_x, V_x) = (f(t,x,U), V), \tag{1.7}$$

where $U \in S_\kappa^\nu$, $\forall V \in S_{\nu,0}^\kappa = \{v \in S_\kappa^\nu : v(0) = v(1) = 0\}$. The initial value $U(x,0)$ is usually determined by L^2-projection of $u_0(x)$ into S_κ^ν. That is, $U(x,0) = \sum_{j=1}^N C_j\psi_j(x)$, where C_j is decided by using the least squares method. Substituting $U(t,x) = \sum_{j=1}^N \psi_j(x)\phi_j(t)$ and $V = \psi_i(x)$ in (1.7) yields

$$\sum_{j=1}^N (\psi_j(x), \psi_i(x))\phi_j'(t) + \sum_{j=1}^N (a(x)\psi_j'(x), \psi_i'(x))\phi_j(t)$$

$$= (f(t,x,\sum_{j=1}^N \psi_j(x)\phi_j(t)), \psi_i(x)), \qquad i = 1, \cdots, N, \qquad t > 0.$$

The resulting system is a nonlinear initial-value problem, which has the form

$$A\vec{\phi}' = \vec{F}(t, \vec{\phi}), \tag{1.8}$$

where $A = [(\psi_j(x), \psi_i(x))]$, $\vec{\phi} = (\phi_1, \cdots, \phi_N)^T$ and $\vec{F}$ is a vector of nonlinear functions. Note that, unlike (1.5), equation (1.8) gives $\vec{\phi}'$ implicitly. This places a restriction on the ODE software we can use.

(2) *Collocation*

The collocation method is a special type of finite element method. In this thesis our code is based on EPDCOL [KM91], in which $\nu = 2$ and B-spline polynomials [B77] are used as the basis of the approximate subspace. Therefore we will only discuss this approach. However, other subspaces can also be used, such as monomial splines [KMN96, N95], etc. We know that the dimension of the subspace is $N = (\kappa - 1)NINT + 2$. Since we have two boundary conditions, we choose the other collocation points as

$$\Lambda = \{\xi_l\}_{l=1}^{N-2} = \{x_i + h_i\rho_r; \quad i = 0, \cdots, NINT - 1; \quad r = 1, \cdots, (\kappa - 1)\},$$

where $\{\rho_r\}_{r=1}^{\kappa-1}$ is the set of zeros of the degree $(\kappa-1)$ Legendre polynomial on the interval $[0,1]$,

$$h_i = x_i - x_{i-1}.$$

This is a common choice for the collocation points, and in this case, it is referred to as Gaussian Collocation. The approximate solution is chosen as

$$U(t,x) = \sum_{j=1}^{N} \psi_j(x)\phi_j(t),$$

where the $\psi_j(x)$ are chosen as elements of a B-spline basis. By substituting this into (1.1)-(1.4), we obtain

$$\sum_{i=1}^{N} \psi_i(\xi_l)\phi_i'(t) = f(t,\xi_l,U(t,\xi_l),U_x(t,\xi_l),U_{xx}(t,\xi_l)), \qquad l = 1,\cdots,N-2. \tag{1.9}$$

Now we have $N-2$ ODEs coupled with two boundary condition, which gives $N-2$ equations in $N-2$ unknowns.

There are two approaches to applying the boundary conditions. The first one is to form an ODE corresponding to the point $x=0$ by differentiating the boundary condition (1.3) with respect to t, which gives

$$\frac{\partial b_L}{\partial t} + \frac{\partial b_L}{\partial u}U_t + \frac{\partial b_L}{\partial u_x}(U_x)_t = 0.$$

Similarly, by differentiating the other boundary condition (1.4), we get another ODE corresponding to the point $x=1$. Combining the boundary condition equations with (1.9) yields a system of N time dependent ordinary differential equations which have the form

$$A\frac{d\vec{\phi}}{dt}(t) = \vec{g}(t,\vec{\phi}),$$

where $\vec{\phi} = (\phi_1,\cdots,\phi_N)^T$. The matrices A are banded. PDECOL handled these as simple banded matrices, while EPDCOL took advantage of the fact that they are

almost block diagonal [DFK83].

An alternative approach does not differentiate the boundary conditions. Rather, we apply collocation to the undifferentiated boundary conditions to give 2 more algebraic equations, resulting in a differential algebraic system of equations (DAE) [P82].

Chapter 2

Some Adaptive Mesh Techniques for 1-D PDE

During the last twenty years, several researchers have studied adaptive mesh methods for parabolic PDEs. Most of them have implemented their mesh algorithm in a MOL code. As mentioned before, adaptive grid technique provide a far more efficient and accurate way to deal with problems involving large solution variation than the traditional MOL approach.

Roughly speaking, there exist at least three approaches. The first approach is called r-refinement, in which the grid movements are time-dependent funtions. After spatial discretization, the MOL gives a system of ODEs for the solution, which are coupled with a system of ODEs controlling the mesh selection. The resulting systems are then solved simultaneously. The number of grids points is fixed. A well-known example is the moving finite element method presented by Miller and Miller ([MM81], [M81]) and later extended by several authors (see, e.g., Adjerid and Flaherty [AF86a], [AF86b]). Another example is the technique suggested by White ([W82]), based on the equidistribution principle (which will be introduced shortly). It was extended by Huang, Ren and Russell ([HRR94a], [HRR94b]).

The second approach is h-refinement. The mesh is refined or coarsened after one or several time steps, and the mesh selection is not coupled with the solution computation. An interpolation after regridding is necessary to generate the initial values for the next step. Some recent contributions in this area are Sanz-Serna and Christie ([SC86]), Adjerid et. al. ([AFMW92],[AFW93]).

The third approach is called p-refinement. It fixes the mesh and varies the degree of piecewise polynomial. When dealing with time-dependent problems, this approach is often combined with h-refinement and then called h-p refinement. Some recent contributions in this area are Gui and Babuska ([GB86a], [GB86b], [GB86c]).

In this thesis we will not discuss p-refinement, but we will review some recent adaptive techniques using h-refinement and r-refinement in the rest of Chapter 2.

2.1 H-refinement

Before we discuss the h-refinement strategy, we present the equidistribution principle (EP).

White [W79] first introduced this idea for a two-point boundary-value problem

$$u' = f(u, x), \qquad x \in (0, 1), \tag{2.1}$$

$$b(u(0), u(1)) = 0, \tag{2.2}$$

where u, f, b are n-vectors. We then consider the transformation of the form

$$\xi(x) = \frac{1}{\theta} \int_0^x M(\tilde{x}) d\tilde{x}, \tag{2.3}$$

where $M(x)$ is some monitor function that provides a measure of the computational error, and

$$\theta = \int_0^1 M(\tilde{x})d\tilde{x}. \tag{2.4}$$

Two versions of $M(x)$ were used by White. One was arc length, and the other was local truncation error. When arc length was used, $M(x) = (1 + \|du/dx\|^2)^{1/2}$ and (2.3), (2.4) became

$$\xi(x) = \frac{1}{\theta}\int_0^x (1 + \|du/d\tilde{x}\|^2)^{1/2}d\tilde{x}$$

and

$$\theta = \int_0^1 (1 + \|du/d\tilde{x}\|^2)^{1/2}d\tilde{x}.$$

Clearly, $\xi(x)$ and θ must satisfy the boundary-value problem

$$\frac{d\xi}{dx} = (1 + \|du/dx\|^2)^{1/2}/\theta, \qquad x \in (0,1), \tag{2.5}$$

$$\frac{d\theta}{dx} = 0, \qquad x \in (0,1) \tag{2.6}$$

$$\xi(0) = 0, \qquad \xi(1) = 1. \tag{2.7}$$

We can see that the combined problem consists of (2.1) (2.2) (2.5), (2.6) and (2.7). Let x_i, $i = 0, \cdots, NINT$, be a set of mesh points such that

$$0 = x_0 < x_1 < x_2 < \cdots < x_{NINT} = 1.$$

We want to determine x_i, $i = 0, \cdots, NINT$ such that

$$\xi(x_i) - \xi(x_{i-1}) = \frac{1}{NINT}, \qquad i = 1, \cdots, NINT, \tag{2.8}$$

in order to equidistribute in the ξ domain. In [W79] Equations (2.1), (2.5), (2.6) are discretized by using a central difference method. If the local truncation error is chosen

as the monitor function, the basic idea is the same. Obviously, EP transforms the physical coordinate x to a computational one ξ, which is always some error measure. Then ξ is equidistributed to obtain the new mesh. EP is widely used since it has been shown to serve as a good mesh-selection strategy.

Sanz-Serna and Christie [SC86] extended the idea of White [W79] to the parabolic PDE in (1.1)-(1.4), given here for convenience

$$u_t = f(t, x, u, u_x, u_{xx}), \quad 0 < x < 1, \quad t > 0, \tag{2.9}$$

$$u(0, x) = u_0(x), \quad 0 < x < 1, \tag{2.10}$$

$$b_L(t, u(t, 0), u_x(t, 0)) = 0, \quad t > 0, \tag{2.11}$$

$$b_R(t, u(t, 1), u_x(t, 1)) = 0, \quad t > 0. \tag{2.12}$$

In the rest of Chapter 2, we will discuss this problem. Since this problem is a time-dependent one, we can see that the computational frame ξ now depends on not only x but also t. Therefore, we use the new pair of coordinates (ξ, T) and the transformation from (x, t) to (ξ, T) is written as

$$\xi(x, t) = \frac{1}{\theta(t)} \int_0^x M(\tilde{x}, t) d\tilde{x}, \tag{2.13}$$

$$T = t, \tag{2.14}$$

where $M(x, t)$ is some monitor function that provides a measure of the computational error, and

$$\theta(t) = \int_0^1 M(\tilde{x}, t) d\tilde{x}. \tag{2.15}$$

From (2.13) and (2.14), it is clear that x can be expressed as a function of ξ and T. Let x_i^n and θ_n approximate to $x(\xi_i, T_n)$ and $\theta(T_n)$ respectively, where $i = 0, 1, \cdots, NINT$

and $n \geq 0$. Then let u_i^n be the approximation of $u(x(\xi_i, T_n), T_n)$. By using the arc length, $\theta(t) = \int_0^1 (1 + |u_x|^2)^{1/2} dx$ as the error measure, Sanz-Serna and Christie approximated θ_n by a Riemann sum and finite differences to obtain

$$\theta_n \cong \sum_{i=1}^{NINT} \sqrt{(x_i^n - x_{i-1}^n)^2 + (u_i^n - u_{i-1}^n)^2}. \tag{2.16}$$

Suppose that u_i^n and x_i^n are known. Let us consider advancing the time step from T_n to T_{n+1}. By first differencing the spatial variable and using an implicit Euler method on the time step on (2.9) with a fixed grid, Sans-Serna and Christie predicted the value

$$\tilde{u}_i^{n+1} \cong u(x_i^n, T_{n+1}).$$

Then new node positions x_i^{n+1} were then chosen by applying EP using arc length as the monitor function. After regridding, the solution u_i^{n+1} on the new gird was obtained from $\tilde{u}_i^{n+1}$ by means of cubic interpolation. Thus, the meshes were regridded after every time step.

This approach is a typical h-refinement. It is straightforword to program. But the shortcomings are also obviously. Firstly, since there is no nature interpolant associated with the finite difference spatial discretization, an extra computation associated with obtaining an interpolant is required, while finite element methods have a nature interpolant. Secondly, it is difficult to extend to a higher order.

For the finite element method, an h-refinement method is derived by Adjerid et. al. [AFMW92, AFW93]. The Galerkin method is used with a p^{th}-degree piecewise polynomial basis. Then an *a posteriori* error estimate is obtained from the solution of a local parabolic problem using a $p + 1^{th}$-degree piecewise polynomial basis. After the error estimate is obtained, a regridding algorithm similar to that at [AF86b] (which will be discussed later) is empolyed. To the best of our knowledge, this is the only approach available in the literature in which a high order method is obtained.

Therefore, we will give a detailed discussion of this approach in Chapter 3.

2.2 R-refinement

Huang, Ren and Russell [HRR94a, HRR94b] derived some moving mesh PDEs (MMPDE) methods based on the EP. Their approach uses standard r-refinement and spatial discretizations based on finite differences. We have x as a function of ξ and T, i.e. $x(\xi, T)$, and in (2.15) θ as a function of T, i.e. $\theta(T)$. Then from (2.13) we have

$$\xi\theta(T) = \int_0^{x(\xi,T)} M(\tilde{x}, t)d\tilde{x}. \tag{2.17}$$

Differentiating (2.17) with respect to ξ twice, they obtained a differential form of the EP,

$$\frac{\partial}{\partial\xi}\{M(x(\xi, T), t)\frac{\partial}{\partial\xi}x(\xi, T)\} = 0. \tag{2.18}$$

Then they required 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

$$\frac{\partial}{\partial\xi}\{M(x(\xi, T + \tau), t + \tau)\frac{\partial}{\partial\xi}x(\xi, T + \tau)\} = 0. \tag{2.19}$$

Equation (2.19) introduces a relaxation time, τ, for the mesh to satisfy the EP. By using the Taylor expansions of $\frac{\partial}{\partial\xi}x(\xi, T + \tau)$ and $M(x(\xi, T + \tau), t + \tau)$ at $x(\xi, T)$ in (2.19) and dropping higher-order terms, they obtained the so-called MMPDE1 of the form

$$\frac{\partial}{\partial\xi}(M\frac{\partial\dot{x}}{\partial\xi}) + \frac{\partial}{\partial\xi}(\frac{\partial M}{\partial\xi}\dot{x}) = -\frac{\partial}{\partial\xi}(\frac{\partial M}{\partial t}\frac{\partial x}{\partial\xi}) - \frac{1}{\tau}\frac{\partial}{\partial\xi}(M\frac{\partial x}{\partial\xi}). \tag{MMPDE1}$$

where $\dot{x} = x_T$. In [HRR94a, HRR94b], the authors took $M(x, t)$ as the arclength monitor function

$$M = \sqrt{1 + u_x^2}.$$

By using central difference methods in space, the discrete approximation of MMPDE1 is given on the uniform computational mesh $\xi_i = \frac{i}{NINT}$, $i = 1, \cdots, NINT - 1$. However, there was some difficulty in computing the basic terms, $\partial M/\partial t$, so the authors dropped the term $-\frac{\partial}{\partial \xi}(\frac{\partial x}{\partial \xi}\frac{\partial}{\partial t}M)$ in MMPDE1. This gives the simplification MMPDE2,

$$\frac{\partial^2}{\partial \xi^2}(M\dot{x}) = -\frac{1}{\tau}\frac{\partial}{\partial \xi}(M\frac{\partial x}{\partial \xi}) \qquad \text{(MMPDE2)}$$

Again using central differencing, its discrete approximation is given by

$$\frac{1}{(1/NINT)^2}[M_{i+1}\dot{x}_{i+1} - 2M_i\dot{x}_i + M_{i-1}\dot{x}_{i-1}] = -\frac{E_i}{\tau}. \qquad (2.20)$$

Here, E_i is the discrete approximation of $\frac{\partial}{\partial \xi}(M\frac{\partial x}{\partial \xi})$ at $\xi = \xi_i$ given by

$$E_i = \frac{M_{i+1} + M_i}{2(\frac{1}{NINT})^2}(x_{i+1} - x_i) - \frac{M_i + M_{i-1}}{2(\frac{1}{NINT})^2}(x_i - x_{i-1}). \qquad (2.21)$$

In oder to obtain this, $M(x,t)$ was discretized with central differences at interior nodes and with one-sided two-point differences at boundary nodes. They then coupled (2.20) with the discretized form of the PDE (2.9) obtained by the central difference method,

$$\dot{u}_i - \frac{(u_{i+1} - u_{i-1})}{(x_{i+1} - x_{i-1})}\dot{x}_i = f_i, \qquad i = 1, 2, \cdots, NINT - 1, \qquad (2.22)$$

where $\dot{u}_i$ denotes the approximation for $u_T(\xi_i, T)$ and f_i denotes the discrete approximation to the operator f at $\xi = \xi_i$ using centred finite differences. Finally the resulting ODE system was solved by using the stiff ODE solver DASSL, although no extra algebraic constraints are invovled.

In summary, in [HRR94a, HRR94b], several MMPDEs are obtained. There are two ways to derive such equations. The first one is to differentiate the EP equation with respect to t. The second one gives a relaxation time τ and requests that the EP is satisfied at time $t + \tau$ instead of at t. Then the MMPDE is coupled with the discretized physical PDE. This approach does not need the interpolation which is always necessary for h-refinement. Another advantage is that it computes the solutions at

the mesh points and the locations of the mesh points at the same time. Therefore, it does not need to restart and is thus much more efficient than h-refinement. However, there are three disadvantages. Firstly, there is no general analysis for the error measure and stability. Secondly, an r-refinement method fixes the value of $NINT$, which makes it inflexible in dealing with difficult problems. Thirdly, it is difficult to extend to a high-order method. Since this approach is based on the EP, an error measure is always needed. Huang et. al. succeeded by using arc-length as their error measure only because they used the central difference method, i.e. a second order method. However, for high-order methods, a reasonable error measure is difficult to derive.

In [HR96], a moving collocation method was presented by Huang and Russell. They used a cubic Hermite collocation method for the physical PDE and a central difference method for MMPDE. Surprisingly, computational results indicated third order convergence for the method, which is slower than the traditional (fourth order) cubic collocation on a fixed mesh but much faster than the first order of the commonly used moving finite difference methods. To date, there is no theoretical analysis to confirm those results.

For finite element methods, there are two approaches for r-refinement: (i) move the mesh so that some estimate of the spatial error is equidistributed [AF86a, AF86b]; (ii) move the mesh so that the residual $\|\dot{U} - L(U)\|$ is minimized by a least squares method [MM81, M81], where U is a piecewise linear approximation to u and $L(U) = f(t, x, U, U_x, U_{xx})$.

The first paper on the moving finite element method (MFE) was written by Miller et. al. [MM81, M81]. They introduced a mesh

$$\Pi(t) := \{0 = x_0(t) < x_1(t) < \cdots < x_{NINT-1}(t) < x_{NINT}(t) = 1\} \tag{2.23}$$

on $(0,1)$ and selected the approximation of the solution u as a piecewise linear function

$$U(x,t) = \sum_{i=0}^{NINT} U_i(t)\Phi_i(x,\Pi(t)), \tag{2.24a}$$

where

$$\Phi_i(x,\Pi(t)) = \begin{cases} \frac{x-x_{i-1}(t)}{x_i(t)-x_{i-1}(t)}, & x_{i-1}(t) \le x \le x_i(t), \\ \frac{x_{i+1}(t)-x}{x_{i+1}(t)-x_i(t)}, & x_i(t) \le x \le x_{i+1}(t), \\ 0, & \text{otherwise}, \end{cases} \tag{2.24b}$$

where $U_i(t) \cong u(x_i(t),t)$. It is easy to see that the derivative U_t that has the form

$$U_t = \sum_{i=0}^{NINT} \{\dot{U}_i(t)\Phi_i(x,\Pi(t)) + \sum_{j=i-1}^{i+1} U_i(t)\frac{\partial \Phi_i}{\partial x_j}\dot{x}_j(t)\},$$

where $\dot{U}_i = \partial U_i/\partial t$ and $\dot{x}_i = \partial x_i/\partial t$. After some simple calculations, they obtained

$$U_t = \sum_{i=0}^{NINT} [\dot{U}_i(t)\Phi_i(x,\Pi(t)) + \dot{x}_i(t)\beta_i(x,\Pi(t))], \tag{2.25}$$

where

$$\beta_i(x,\Pi(t)) = \begin{cases} -\frac{(U_i(t)-U_{i-1}(t))(x-x_{i-1}(t))}{(x_i(t)-x_{i-1}(t))^2}, & x_{i-1}(t) \le x \le x_i(t), \\ -\frac{(U_{i+1}(t)-U_i(t))(x_{i+1}(t)-x)}{(x_{i+1}(t)-x_i(t))^2}, & x_i(t) \le x \le x_{i+1}(t), \\ 0, & \text{otherwise}. \end{cases} \tag{2.26}$$

To minimum the L^2-norm of the residual

$$\|R(U)\| = \|U_t - L(U)\|$$

$$= \sum_{i=0}^{NINT}\sum_{j=0}^{NINT} [(\Phi_i,\Phi_j)\dot{U}_i\dot{U}_j + (\beta_i,\beta_j)\dot{x}_i\dot{x}_j + (L(U),L(U)) - (\beta_i,L(U))\dot{x}_i$$

$$-(\beta_j,L(U))\dot{x}_j + (\Phi_i,\beta_j)\dot{U}_i\dot{x}_j + (\beta_i,\Phi_j)\dot{x}_i\dot{U}_j - (\Phi_i,L(U))\dot{U}_i - (\Phi_j,L(U))\dot{U}_j],$$

they applied the least square method to $\|R(U)\|$ by differentiating $\|R(U)\|$ with respect to $\dot{U}_i$ and $\dot{x}_i$. They then obtained a system of $2(NINT-1)$ equations in the $2(NINT-1)$ unknowns U_i, x_i:

$$\sum_{j=0}^{NINT} [(\Phi_i, \Phi_j)\dot{U}_j + (\Phi_i, \beta_j)\dot{x}_j] = (\Phi_i, L(U)), \qquad 1 \le i \le NINT-1, \tag{2.27a}$$

$$\sum_{j=0}^{NINT} [(\beta_i, \Phi_j)\dot{U}_j + (\beta_i, \beta_j)\dot{x}_j] = (\beta_i, L(U)), \qquad 1 \le i \le NINT-1, \tag{2.27b}$$

where $(\cdot,\cdot)$ denoted the usual inner product. Then this ODE system was integrated to obtain the linear approximate solution. Here the initial conditions $U_i(0)$ and $x_i(0)$ are based on the uniform grids, i.e. $x_i(0) = i/NINT$ and $U_i(0) = u_0(i/NINT)$, where $i = 0, 1, \cdots, NINT$.

Adjerid and Flaherty [AF86a, AF86b] obtained a moving finite element method by applying EP to the spatial error in H^1 using the piecewise linear functions (2.24a), (2.24b) as the approximation of the solution u. By introducing certain equations for the error estimate, which were coupled with the mesh equations and the physical equations, they solved these equations simultaneously. First, they obtained equations of the form

$$(V_1, U_t) = (V_1, f(t, x, U, U_x, U_{xx})), \tag{2.28}$$

where V_1 is a set of piecewise linear polynomials based on the partition Π as in (2.23) and $(\cdot,\cdot)$ denotes the usual inner product. They introduced an error estimate, $E(x,t)$, as a set of piecewise quadratic functions, i.e.,

$$E(x,t) = \sum_{i=1}^{NINT} E_i(t)\Psi_i(x, \Pi(t)), \tag{2.29a}$$

where

$$\Psi_i(x, \Pi(t)) = \begin{cases} \frac{4[x - x_{i-1}(t)][x_i(t) - x]}{[x_i(t) - x_{i-1}(t)]^2}, & x_{i-1}(t) < x < x_i(t), \\ 0, & \text{otherwise}, \end{cases} \tag{2.29b}$$

and $E_i(t)$ is the value of the estimated error, $E(x,t)$, at the corresponding nodal positions. The next step is to solve the ODE

$$(V_2, U_t + E_t) = (V_2, f(t, x, U + E, U_x + E_x, U_{xx} + E_{xx})), \tag{2.30}$$

where V_2 consisted of piecewise quadratics which varnished at the node points. This idea was first introduced by Bieterman and Babuska. In [BB82a, BB82b], they proved that the estimated $E(x,t)$ converged to the exact error when $NINT \to \infty$.

The mesh points are determined by solving the ODE system

$$\dot{x}_i(t) - \dot{x}_{i-1}(t) = -\lambda(\|E_i\Psi_i\|_1 - \bar{E}(t)), \qquad i = 1, 2, \cdots, NINT, \tag{2.31a}$$

where λ is a positive constant (discussed later) and

$$\bar{E}(t) = \frac{1}{NINT}\|E(x,t)\|_1 := \frac{1}{NINT}\{\int_0^1 [(E(x,t))^2 + (E_x(x,t))^2]dx\}^{1/2}. \tag{2.31b}$$

Here $\|E_i\Psi_i\|_1$ is the local error in H^1 on $(x_{i-1}(t), x_i(t))$ and $\bar{E}(t)$ is the average error in H^1 over the whole problem interval $[0,1]$. If $\|E_i\Psi_i\|_1$ is larger than $\bar{E}$ then the points $x_i(t)$ and $x_{i-1}(t)$ are moved together. Similarly, when $\|E_i\Psi_i\|_1$ is smaller than $\bar{E}$ the points $x_i(t)$ and $x_{i-1}(t)$ are moved apart. Since $\bar{E}$ is usually difficult to calculate, Adjerid et. al. eliminated $\bar{E}$ (2.31a) on two neighbouring intervals. This gives

$$\dot{x}_{i+1} - 2\dot{x}_i + \dot{x}_{i-1} = -\lambda(\|E_{i+1}\Psi_{i+1}\|_1 - \|E_i\Psi_i\|_1), \qquad i = 1, 2, \cdots, NINT - 1, \;\; t > 0. \tag{2.32}$$

Then the differential algebraic system solver DASSL is used to solve the ODE system, (2.28), (2.30), (2.32), to obtain the required fully discretized solution.

These were the first papers describing an adaptive strategy which was based on an error estimate (proved to converge to the exact error), and for which the equidistribution strategy was proved to be stable [CFL86]. Unfortunately, sometimes the solutions are sensitive to the user supplied parameter λ (see, e.g. example 3 [AF86a],

the reaction-diffusion model). However, this approach is still very interesting since it gives the possibility of obtaining an error estimate for a high-order method (Adjerid et. al. later derived a high-order adaptive method [AFW93]). Since it is a finite element method, there is no difficulty with interpolation. Therefore, it is possible to extend this approach to an h-refinement method. In fact our work follows this idea.

Before we finish discussing r-refinement, there is one thing which must be mentioned. Since the locations of the mesh and the solution at the grid points are obtained simultaneously, sometimes the grid points can cross or move out of the domain. To prevent this, a spatial smoothing technique is necessary, since generally, the monitor function is not smooth. Therefore, certain terms, which give some artifical diffusion, are added to smooth the monitor function, M. For example, in [HR97], Huang and Russell give

$$\tilde{M} - \lambda^{-2}\tilde{M}_{\xi\xi} = M,$$

where λ is a large positive number. Then they require that $\tilde{M}$, instead of M, satisfies the MMPDE. Computational results show that, by smoothing M, we can prevent the mesh crossing.

In addition to h-refinement and r-refinement, there exists an intermediate approach. Verwer, Blom and Sanz-Serna ([BSV88], [VBS89]) derived this technique. First, through the same co-ordinate transformation as the one in [SC86], they obtained (ξ, T) and wrote equation (2.9) as

$$u_t = u_T - u_x x_T = f(t, x, u, u_x, u_{xx}), \quad 0 < \xi < 1,\ T > 0. \tag{2.33}$$

They then multiply (2.33) by $\partial x/\partial\xi = x_\xi$ to obtain

$$x_\xi u_T - u_\xi x_T = x_\xi f(t, x, u, u_x, u_{xx}), \quad 0 < \xi < 1,\ T > 0, \tag{2.34}$$

and follow this by applying standard central differencing on the uniform ξ-grid $\{\xi_i = ih,\ 0 \le i \le NINT,\ h = 1/NINT\}$. They then solve the resultant ODE system using

the so-called Lagrangian time-stepping scheme

$$(\alpha(x_{i+1}^{n+1} - x_{i-1}^{n+1}) + (1-\alpha)(x_{i+1}^{n} - x_{i-1}^{n}))(\frac{u_i^{n+1} - u_i^n}{t_{n+1} - t_n})$$

$$-(\alpha(u_{i+1}^{n+1} - u_{i-1}^{n+1}) + (1-\alpha)(u_{i+1}^{n} - u_{i-1}^{n}))(\frac{x_i^{n+1} - x_i^n}{t_{n+1} - t_n})$$

$$= \alpha(x_{i+1}^{n+1} - x_{i-1}^{n+1}) f_{h,i}(u^{n+1}) + (1-\alpha)(x_{i+1}^{n} - x_{i-1}^{n}) f_{h,i}(u^n), \tag{2.35}$$

where $0 \leq \alpha \leq 1$ and $i = 1, \cdots, NINT - 1$, u_i^n is the discrete approximation to the value $u(x_i^n, t_n)$ and $f_{h,i}$ is the approximation $f(t, x, u, u_x, u_{xx})$ obtained by means of central differences. As usual, t_n and t_{n+1} are consecutive time-levels. It is easy to see that if $\alpha = 1$, (2.35) becomes the implicit formula

$$\frac{u_i^{n+1} - u_i^n}{t_{n+1} - t_n} = f_{h,i}(u^{n+1}).$$

Also we can see that if $\alpha = 0$, (2.35) becomes the explicit formula

$$\frac{u_i^{n+1} - u_i^n}{t_{n+1} - t_n} = f_{h,i}(u^n),$$

and if $\alpha = 1/2$, it becomes the Bonnerot-Jamet-Crank-Nicolson (BJCN) scheme.

Given a mesh at the nth time-level and the corresponding numerical solution, the mesh and the solutions at the $(n+1)$th time-level are obtained in two steps. The first one is the *grid prediction*. First an implicit Euler step is performed with the spatial grid fixed (i.e., (2.35) is applied with $\alpha = 1$ and $x_i^{n+1} = x_i^n$). In ([BSV88], [VBS89]) the monitor function is chosen as

$$M(\xi, t) = \beta + \sqrt{|\tilde{u}_{xx}(\xi, t)|} \tag{2.36}$$

where β is chosen as 1. Then in the subinterval (x_{i-1}, x_i), the error estimate $E_i(t)$ given by

$$(x_i - x_{i-1}) M(\frac{x_i + x_{i-1}}{2}, t) = E_i, \quad 1 \leq i \leq NINT. \tag{2.37}$$

By applying EP to E_i, they obtained the new mesh. In the second step, *integration step*, they computed the approximations u_i^{n+1} according to (2.35) with $\alpha = \frac{1}{2}$, i.e. changed the formula to the BJCN scheme. Since $\{x_j^n\}_{j=0}^{NINT}$, $\{u_j^n\}_{j=0}^{NINT}$ and $\{x_j^{n+1}\}_{j=0}^{NINT}$ are now known, we can obtain $\{u_j^{n+1}\}_{j=0}^{NINT}$ using the BJCN scheme.

This approach has two advantages. First, the mesh-selecting algorithm is a typical h-refinement. Therefore, the mesh will not cross or move out of the domain. Second, it does not need interpolation (By changing the formula to the BJCN scheme). This aspects is the same as in r-refinement. Therefore, it takes advantage of some aspects of both h-refinement and r-refinement. But the disadvantages are that it relies on the accuracy of *grid prediction* too much and it is difficult to generate a high-order formula.

2.3 Available Adaptive Software for 1-D Parabolic PDE

Adaptive software for ordinary differential equations has existed for a long time; however, the situation is much different for partial differential equations. In order to get a summary of available software, one can access the Guide to Available Mathematical Software (GAMS) resource [BHK85]. However, we find that there is only one commercial library package (NAG) which includes two adaptive subroutines that solve problems of the class defined by (1.1)-(1.4), and no public package is available.

The routines listed in GAMS are:

- **D03PPF/D03PRF (NAG)** which are MOL packages. The spatial variable is discretized using the central difference method. The monitor function is chosen as the second space derivative of the solution. The mesh is chosen

to equidistribute the integral of the monitor function over the domain. Both packages are modified versions of SPRINT [BDF89].

The following packages, which are not listed in GAMS or netlib, are MOL packages and are only available directly from their authors:

- **MOVCOL** [HR96, HR99] which uses a cubic Hermite collocation for the physical PDEs and a central difference method for the PDE which determines the moving mesh. This was discussed in r-refinement. The resulting ODE system is solved using DASSL [P82].

- **PDEFRONT** [AFMW92] which can deal with one- or two-dimension parabolic PDEs. A Galerkin method is used. An *a posteriori* estimation of the discretization errors is obtained using polynomials having a degree one higher than that for the solution. The refinement algorithm is of the type hrp. In contrast to the previously mentioned packages which are written in Fortran, this is written in a combination of C and MAPLE for Sun workstations.

Chapter 3

Adaptive Collocation Solution and Error Estimates

3.1 A Posteriori Error Estimates

Adjerid et. al. presented a high-order adaptive MOL package, *pdefront* [AFMW92], which applies EP to the computed *a posteriori* error estimate of the spatial discretization error. The Galerkin method is used to seek a piecewise polynomial approximation of degree p. The error estimates are obtained by solving a system of local parabolic problems or local elliptic ones with $(p+1)$th piecewise polynomial basis. This error-estimation strategy has been proved to converge to the exact error for linear, one-dimensional, Dirichlet problems [AFW93]. Computational results show that this error evaluation converges to the true error even for nonlinear problems. However, to our knowledge, there is no analysis to confirm this. Since our code is based on the ideas considered in [AFW93], we now give a detailed introduction to this paper.

Adjerid et. al. consider an *a posteriori* estimation procedure for a linear Dirichlet problem

$$u_t + Lu = f(x,t), \qquad 0 < x < 1, \;\; t > 0, \tag{3.1a}$$

$$Lu = -[D(x)u_x]_x + q(x)u, \tag{3.1b}$$

$$u(x,0) = u_0(x), \qquad 0 \le x \le 1, \tag{3.1c}$$

$$u(0,t) = u(1,t) = 0, \qquad t > 0, \tag{3.1d}$$

where $D(x) > 0$ and $q(x) \ge 0$ for $0 \le x \le 1$. They first define mesh

$$\Delta_N := \{0 = x_0 < x_1 < \cdots < x_{N-1} < x_N = 1\}$$

Then let $A(v,u) = \int_0^1 [v_x D(x) u_x + vq(x)u]dx$, using a Galerkin's method, they compute an approximate solution $U \in S_0^{N,p}$ (to be defined shortly), which satisfies

$$(V, U_t) + A(V, U) = (V, f), \qquad t > 0, \tag{3.2a}$$

$$A(V, U) = A(V, u_0), \qquad t = 0, \qquad \text{for all } V \in S_0^{N,p}. \tag{3.2b}$$

The approximating subspace is given by

$$S_0^{N,p} := \{W(x) \mid W \in H_0^1, \quad W \in P_p \text{ for } x \in (x_{j-1}, x_j), j = 1, 2, \cdots, N\}, \tag{3.3}$$

where P_p consists of polynomials whose degree is less than or equal to $p > 0$ in x, and H_0^1 consists of functions which are continuous for $x \in (0,1)$ and satisfy the homogeneous boundary conditions. The L^2 inner products is

$$(v,u) = \int_0^1 v(x,t)u(x,t)dx, \tag{3.4}$$

The error

$$e(x,t) = u(x,t) - U(x,t) \tag{3.5}$$

is approximated by $E(x,t) \in \hat{S}_0^{N,p+1}$, where $\hat{S}_0^{N,p+1} = \bigcup_{j=1}^N \hat{S}_{0,j}^{p+1}$ and

$$\hat{S}_{0,j}^{p+1} := \{W(x) \mid W \in P_{p+1} \text{ for } x \in (x_{j-1}, x_j), \ W \equiv 0, \ \text{otherwise}\}, \tag{3.6}$$

$$j = 1, 2, \cdots, N.$$

Then, they require that $U + E$, instead of U, satisfies with (3.2). The estimate E is obtained by solving the local parabolic problems

$$(V, E_t)_j + A_j(V, E) = (V, f)_j - A_j(V, U) - (V, U_t)_j, \qquad t > 0, \tag{3.7a}$$

$$A_j(V, E) = A_j(V, u_0 - U), \qquad t = 0, \qquad \text{for all } V \in \hat{S}^{p+1}_{0,j}, \qquad j = 1, 2, \cdots, N, \tag{3.7b}$$

where the local L^2 and the local energy inner products, respectively, are

$$(v, u)_j = \int_{x_{j-1}}^{x_j} vu dx, \tag{3.8a}$$

$$A_j(v, u) = (v_x, Du_x)_j + (v, qu)_j. \tag{3.8b}$$

By dropping the first term $(V, E_t)_j$, Adjerid et. al. obtain the local elliptic problems

$$A_j(V, E) = (V, f)_j - A_j(V, U) - (V, U_t)_j, \qquad t > 0, \qquad \text{for all } V \in \hat{S}^{p+1}_{0,j} \tag{3.9}$$

$$j = 1, 2, \cdots, N,$$

which are solved to obtain an elliptic error estimate.

Adjerid et. al. proved the convergence of the parabolic error estimation procedure; the result is given by the following theorem

Theorem 1 *Let u, $U \in S^{N,p}_0$, and $E \in \hat{S}^{N,p+1}_0$ be solutions of (3.1), (3.2) and (3.7), respectively, then there exist positive constants C and δ such that*

$$\|e(\cdot, t)\|_1^2 = \|u(\cdot, t) - U(\cdot, t)\|_1^2 = \|E(\cdot, t)\|_1^2 + \epsilon, \qquad t > \delta > 0, \tag{3.10a}$$

where

$$|\epsilon| \leq C(u)h^{2p+1}, \tag{3.10b}$$

$$\|u\|_1 = \sqrt{\sum_1^N ((u,u)_j + (u_x,u_x)_j)}. \tag{3.10c}$$

The convergence analysis for the elliptic error procedure (3.9) is similar to the analysis for the parabolic precedure (3.7). Adjerid et. al. also proved a similar result for (3.9).

3.2 An Adaptive Collocation Method

Some computational evidence [N95] indicates that, by using collocation methods based on monomial splines or B-splines, the order of the L^2-norm error is h^{KORD}, where $KORD$ is the order of the piecewise polynomial used and $h = \max_{i=1}^{NINT}(x_i - x_{i-1})$. However, to our knowledge, there is no analysis to confirm this conjecture. Based on the work of Adjerid et. al., we make a similar assumption for the error estimate

Assumption 1 *Using a collocation method based on B-splines to (1.1)-(1.4), suppose that $U \in S_\kappa^\nu$ and $V \in S_{\kappa+1}^\nu$ are two approximate solutions, where*

$$\Pi := \{0 = x_0 < x_1 < \cdots < x_{NINT-1} < x_{NINT} = 1\},$$

$$S_\kappa^\nu := \{W(x) \mid W \in C^{\nu-1} W \in P_\kappa \text{ for } x \in (x_{j-1}, x_j), j = 1, 2, \cdots, NINT\},$$

and P_κ consists of polynomials of degree $\kappa > 0$ in x. Then

$$\|U - V\| \longrightarrow \|U - u\|, \text{ as } h \to 0,$$

and

$$\|U - u\| = O(h^{\kappa+1}),$$

where

$$h = \max_{1 \le j \le NINT} h_j = \max_{1 \le j \le NINT} (x_j - x_{j-1}),$$

$$\|u\| = \sqrt{(u,u)} = \sqrt{\int_0^1 u^2 dx},$$

Assuming we have a good error estimation precedure, the next issue is the spatial remeshing algorithm. Before discussing our remeshing strategy, we review some remeshing strategies implemented in the available software. Roughly speaking, there are four kinds of strategies:

- remesh after a fixed time interval.
- remesh after a fixed number of time steps.
- use the new mesh only if there exists at least one mesh point in which must be moved by more than a certain fraction of the mesh size at that point. The fraction is user supplied.
- give a tolerance on how closely the error measure should be equidistributed, and remesh only if the error measure is beyond the tolerance.

In the thesis, the first of these strategies is chosen to be our remeshing strategy. Based on Assumption I, we use h-refinement. In addition, however, in some situations, we will, as an alternate, choose to double the number of mesh points. Also, in other situations, the orders of the polynomials used in the approximation space will be increased. We have implemented the following Algorithm in our code.

Algorithm

****

Variables: $nint$ (number of subintervals), κ (degree of the B-spline), dt (the fixed time interval before checking the error), eps (the error tolerance), $icount$ (the count of how many remeshings have occurred), $Tend$ (the final value of the time).

****

(1) let $T0 = 0$; $Tout = T0 + dt$; $x(1 : nint + 1)$ is uniform mesh; $icount = 0$;

(2) call PDECOL with $kord = \kappa + 1$, to get the solution ui;

(3) call PDECOL with $kord = \kappa + 2$, to get the solution uii;

(4) $ERROR = \|ui - uii\|$; | in the L^2 norm

(5) if $ERROR \leq eps$ then

    if $Tout < Tend$ then

        $T0 = Tout$; $Tout = Tout + dt$;

        if $Tout > Tend$ then

            $Tout = Tend$;

        endif;

        goto (2); | continue integration

    else

        goto (6); | end of integration

    endif;

else | tolerance not met

    if $icount = 0$ then

        $icount = icount + 1$; $z(1 : nint + 1) = x(1 : nint + 1)$; $zerr = ERROR$;

        remesh and goto (2); | remesh and try again

    else

        if $icount < 3$ then

            if $zerr > ERROR$ then

                $zerr = ERROR$; $z(1 : nint + 1) = x(1 : nint + 1)$;

            endif;

$icount = icount + 1$;
remesh and goto (2); | remesh and try again
else | $icount = 3$
if $zerr < ERROR$ then
$x(1 : nint + 1) = z(1 : nint + 1)$; | mesh changed to the one
| which can get the best error
endif;
icount=0;
if $nint \leq 30$ then
$nint = nint * 2$; half mesh; | double nint and restart
goto (2);
else
if $kord \leq 7$ then
$kord = kord + 1$, | increase kord and restart
goto (2);
else
if $nint \leq 60$ then
$nint = nint * 2$; half mesh; | double nint and restart
goto (2);
else
$kord = kord + 1$; | increase kord and restart
goto (2);
endif;
endif;
endif;
endif;
endif;
(6) end.

Now we will introduce the remeshing strategy based on an *a posteriori* error estimation.

Suppose that the integration has reached the level $t_n = n \cdot dt$, where dt is a fixed time interval. The approximations $ui(x, t_n)$ and $uii(x, t_n)$ have been computed using the B-splines of degree κ and $\kappa + 1$, respectively. Then the error estimate in the L^2 norm

$$\|E(x,t)\| = \|ui - uii\| = \sqrt{\int_0^1 (ui - uii)^2 dx}$$

is computed. If we find that the error $\|E\|$ is bigger than the tolerance EPS, we reject the step, go back to t_n and the remeshing code is employed to yield the new grid points.

In our remeshing code, we compute the L^2-norm error for each subinterval,

$$\|E_i\| = \sqrt{\int_{x_{i-1}}^{x_i} (ui - uii)^2 dx}, \qquad i = 1, \cdots, NINT,$$

where x_i, $i = 0, \cdots, NINT$, is the old mesh.

Then based on the equidistribution principle (EP), our remeshing code tries to find a new mesh, $\tilde{x}_i$, $i = 0, \cdots, NINT$, which should satisfy

$$\left(\sqrt{\int_{\tilde{x}_{i-1}}^{\tilde{x}_i} (ui - uii)^2 dx} \right)^{\frac{1}{KORD}} = \text{constant}, \tag{3.11}$$

where $KORD = \kappa + 1$.

We need to explain why we use an exponent $1/KORD$ in (3.11). Let us review the EP based on local truncation error. In [W79], White assumed that if the local truncation error is

$$e = \left| C h_i^{\kappa+1} u^{(\kappa+1)} \right|,$$

where C is a constant and $h_i = x_i - x_{i-1}$, then the monitor function is chosen as $\left|u^{(\kappa+1)}\right|^{1/(\kappa+1)}$, and one would try to equidistribute $\int_{x_i}^{x_{i+1}} \left|u^{(\kappa+1)}\right|^{1/(\kappa+1)} dx$. That is, one tries to have

$$\int_{x_i}^{x_{i+1}} \left|u^{(\kappa+1)}\right|^{1/(\kappa+1)} dx = \text{constant}. \tag{3.12}$$

It is clear that if (3.12) is satisfied, when $h \to 0$, the local truncation error will be equidistributed. Morever, we can see that $\int_{x_i}^{x_{i+1}} \left|u^{(\kappa+1)}\right|^{1/(\kappa+1)} dx = O(h)$. It means that we should apply the EP to a error measure which is $O(h)$, and the local error will satisfy the EP if the error measure does when $h \to 0$.

Since in our assumption, $\|e\| = \|E\| = O(h^{\kappa+1})$, it is easy to see that (3.11) is $O(h)$ and it is a good choice to employ with the EP.

To employ (3.11), first we assume that in each subinterval the error distribution is uniform. See Figure 3.1.

We now give our algorithm now for the remeshing code.

Remesh Algorithm

****

Variables: $nint$ (number of subintervals), $\{x_i\}_1^{nint+1}$ (the old mesh), $\{y_i\}_1^{nint+1}$ (the new mesh), E_i (the value of $\left(\sqrt{\int_{x_i}^{x_{i+1}}(ui - uii)^2 dx}\right)^{1/KORD}$), S_j (the sum of the first j values of E_i), δ (the average value for all E_i).

****

(1) Given $nint$, x_i, $i = 1, \cdots, nint+1$ and E_i, $i = 1, \cdots, nint$;

(2) $S_1 = E_1$;

(3) $S_i = S_{i-1} + E_i$, $i = 2, \cdots, nint$;

(4) $\delta = S_{nint}/nint$; $y_1 = x_1$; $y_{nint+1} = x_{nint+1}$; $j = 1$; $B = \delta$;

(5) Do **(9)** $i = 2, \cdots, nint$;

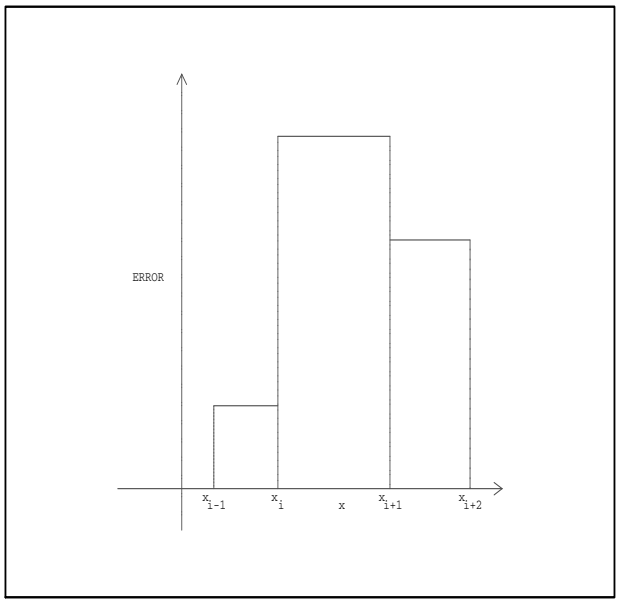

Figure 3.1: Error Distribution

(6) continue;

(7) If $B \geq S_j$ then

j=j+1; goto **(6)**;

else

if $j = 1$ then

$y_i = x_1 + (x_2 - x_1) * B/E_1$;

else

$y_i = x_j + (x_{j+1} - x_j) * (B - E_{j-1})/E_j$;

endif;

endif;

(8) $B = B + \delta$;

(9) continue;

(10) return;

(11) end.

After remeshing, we get the new mesh and go back to last time, t_{n-1}. A cold start is employed in our code although a hot start would be preferable. Suppose that $t_{n-1} \neq 0$, Then the new initial condition comes from our approximation solution $ui(x, t_{n-1})$. If we perform three consecutive remeshings and we still can not meet the error tolerance requested, then the code will decide whether to half the mesh or increase $KORD$ automatically. The criteria are given by

```
if NINT ≤ 30 then
    double NINT
else
    if NINT ≤ 60 then
        if KORD ≤ 7 then
            KORD = KORD + 1
        else
            double NINT
        endif
    else
        KORD = KORD + 1
    endif
endif
```

Since we have 4 meshes, the original one and 3 remeshed ones, the best mesh by which we obtain the smallest error is used for doubling $NINT$ or increasing $KORD$.

Chapter 4

Numerical Experiments

4.1 Testing Methods

There are several (adaptive or non-adaptive) packages which can solve problems of the class of PDEs defined by (1.1)-(1.4). However, since our code uses an adaptive implementation of EPDCOL, a comparison with EPDCOL will show how much improvement in accuracy has been attained by our adaptive strategy. Prior to discussing this comparison, we will clarify two terms. A cold start is the first call to a package for a given problem, assuming no the past information. Therefore, a very small time size is used at the beginning. A hot start is a call to a package which is not the first call for the given problem and a large time size can be achieved at the beginning. A hot start is much more efficient than a cold start. After a mesh redistribution, EPDCOL can only use a cold start, but it can use a hot start when the mesh is fixed. A cold start is performed in our code after a remeshing. Of course this is not efficient. A future package should preferably use a hot start.

We have collected a set of test problems which reflect various difficulties, for which we know the exact solutions. This set includes two parameterized problems which can be used to show the advantages and the shortcomings of our code. All of them have been used as example problems by other authors. Appendix A contains a complete

description of the problems.

Since our code is an adaptive implementation based on EPDCOL, we call it APDCOL, for Adaptive Partial Differential (equations) Collocation.

4.2 Summary of Our Experiments

In this section we provide numerical results to show five things:

- probelm 1, an easy problem, is presented to confirm Assumption 1: by fixing the mesh, the error estimation will converge to the true error as $h \to 0$, if we do not consider the temporal error.
- APDCOL produces smaller error than EPDCOL, and it even succeeds on difficult problems where EPDCOL fails. This will be demonstrated by problem 2 and problem 3.
- dt, a user supplied parameter, plays an important role in the accuracy and efficiency of the computed solution. This is demonstrated in problem 3.
- by computing CPU time and total remesh times for different $NINT$ and $KORD$ in problem 3, it seems to be more efficient, if the error is not equidistributed, to redistribute the mesh even when the error requirement is met.
- Comparison with some with other experimental error measures shows that, (3.11), the one we have chosen to implement, is the best.

4.3 Numerical Results

In our code, we need to compute the error in the L^2 norm. Since the spatial degree of the approximation solution, $ui(x,t)$, is $KORD-1$ and it is $KORD$ for $uii(x,t)$,

$(ui(x,t) - uii(x,t))^2$ is a piecewise polynomial of degree $2KORD$. Therefore, we choose $KORD+1$ Gauss-Legendre points on $[-1,1]$, $\{\rho_r\}_{r=1}^{KORD+1}$, and get

$$\int_{x_i}^{x_{i+1}} (ui - uii)^2 dx = \frac{x_{i+1} - x_i}{2} \sum_{r=1}^{KORD+1} w_r (ui(y_r,t) - uii(y_r,t))^2, \qquad i = 0, \cdots, NINT, \tag{4.1}$$

where $\{w_r\}_{r=1}^{KORD+1}$ are $KORD+1$ Gauss-Legendre weights on $[-1,1]$, and $y_r = \frac{x_i + x_{i+1}}{2} + \frac{(x_{i+1} - x_i)\rho_r}{2}$. We can see that (4.1) gives an exact value for the error estimate. Therefore, to determine the L^2-norm errors between the exact solution $u(x,t)$ and the approximation solution $ui(x,t)$, we use the formula

$$\int_{x_i}^{x_{i+1}} (ui - u)^2 dx = \frac{x_{i+1} - x_i}{2} \sum_{r=1}^{KORD+1} w_r (ui(y_r,t) - u(y_r,t))^2, \qquad i = 0, \cdots, NINT. \tag{4.2}$$

We now present numerical results and discussion for each of the problems contained in Appendix A.

4.3.1 Experimental Confirmation of Assumption 1

Problem 1 was taken from the original PDECOL paper [MS79] and does not require an adaptive mesh to achieve small spatial errors. The reason for choosing this easy problem is to give a computational result which confirms Assumption 1 that the error estimate converges to the true error for fixed mesh. Here we compute a value, θ, defined by

$$\theta = \frac{\|ui - uii\|}{\|ui - u\|},$$

where

$$\|u\| = \sqrt{\int_0^1 u^2 dx}.$$

Table 4.1 gives the value of θ when a uniform fixed mesh is used, for different $NINT$ and $KORD$, with the final time chosen as 0.4. We choose the tolerance for EPDCOL as 10^{-13} because we do not want the spatial errors to be dominated by the temporal error (Recall that EPDCOL does not have a spatial adaptation capability and thus the used tolerance is applied only to the time integration).

	NINT					
	4		8		16	
KORD	$\|ui-u\|$	θ	$\|ui-u\|$	θ	$\|ui-u\|$	θ
4	$3.10 \cdot 10^{-4}$	1.0072	$1.90 \cdot 10^{-5}$	1.0018	$1.19 \cdot 10^{-6}$	1.0005
5	$1.06 \cdot 10^{-5}$	1.0028	$3.29 \cdot 10^{-7}$	1.0007	$1.03 \cdot 10^{-8}$	1.0002
6	$2.82 \cdot 10^{-7}$	1.0016	$4.42 \cdot 10^{-9}$	1.0004	$6.91 \cdot 10^{-11}$	0.9995

Table 4.1: Problem 1: the exact L^2-norm error and the value of θ

Computational results show that the error estimate for each subinterval converges to the true error. We next define the quantities

$$\theta_i = \frac{\|ui - uii\|_i}{\|ui - u\|_i}, \qquad i = 1, \cdots, NINT,$$

where

$$\|u\|_i = \sqrt{\int_{x_i}^{x_{i+1}} u^2 dx}.$$

When $NINT = 8$, $KORD = 5$, the values of θ_i are

$\theta_i =$

1.000040	1.000052	1.000107	1.000765
1.000765	1.000107	1.000052	1.000040

Clearly this example shows that θ_i also tends to 1, for $i = 1, \cdots, NINT$.

4.3.2 Camparison of EPDCOL and APDCOL: Problem 2

Problem 2 is Burgers' equation. Varying the viscosity parameter ϵ reflects varying degrees of difficulty. The results presented in this section use the following parameters:

$$[t_0, t_{end}] = [0, 1], \qquad dt = 0.01,$$

where dt is the fixed time interval before checking the error. For efficiency, we balance the tolerance for the spatial error and the one for the temporal error. That is, we choose the same requested tolerance for both time and space. In the experiment, we choose $TEPS = EPS = 10^{-4}$, where $TEPS$ is to control the temporal error, and EPS is to control the spatial error.

Table 4.2 shows the error when the viscosity parameter $\epsilon = 3 \times 10^{-3}$ and Table 4.3 shows the error when $\epsilon = 10^{-3}$.

			NINT		
			20	30	40
KORD	4	EPDCOL	$2.28 \cdot 10^{-2}$	$7.16 \cdot 10^{-3}$	$2.18 \cdot 10^{-3}$
		APDCOL	$6.82 \cdot 10^{-5}$ ♣	$1.05 \cdot 10^{-4}$	$6.98 \cdot 10^{-5}$
	5	EPDCOL	$6.23 \cdot 10^{-3}$	$1.37 \cdot 10^{-3}$	$6.44 \cdot 10^{-4}$
		APDCOL	$6.87 \cdot 10^{-5}$	$4.07 \cdot 10^{-5}$	$5.71 \cdot 10^{-5}$
	6	EPDCOL	$2.01 \cdot 10^{-3}$	$4.75 \cdot 10^{-4}$	$1.01 \cdot 10^{-4}$
		APDCOL	$5.93 \cdot 10^{-5}$	$6.10 \cdot 10^{-5}$	$6.73 \cdot 10^{-5}$

Table 4.2: Problem 2: L^2-norm errors with $\epsilon = 3 \times 10^{-3}$

Here ♣ means that the mesh algorithm chose to double $NINT$. In both tables, APDCOL gives more accurate solutions than EPDCOL. When the problems are more difficult, the error is dominated by the spatial error when EPDCOL is employed, while the error tolerances are met by remeshing and doubling $NINT$ in APDCOL. As expected, the cost for APDCOL is much more than that for EPDCOL. That is

			NINT		
			20	30	40
KORD	4	EPDCOL	0.13	$7.22 \cdot 10^{-2}$	$4.86 \cdot 10^{-2}$
		APDCOL	$1.48 \cdot 10^{-4}$ ♣	$1.72 \cdot 10^{-4}$	$1.40 \cdot 10^{-4}$
	5	EPDCOL	$8.58 \cdot 10^{-2}$	$5.35 \cdot 10^{-2}$	$2.48 \cdot 10^{-2}$
		APDCOL	$8.41 \cdot 10^{-5}$	$7.11 \cdot 10^{-5}$	$3.35 \cdot 10^{-5}$
	6	EPDCOL	$7.40 \cdot 10^{-2}$	$2.82 \cdot 10^{-2}$	$1.57 \cdot 10^{-2}$
		APDCOL	$7.03 \cdot 10^{-5}$	$2.20 \cdot 10^{-5}$	$1.74 \cdot 10^{-5}$

Table 4.3: Problem 2: L^2-norm errors with $\epsilon = 10^{-3}$

mainly because of the cold starts that are performed after remeshing.

4.3.3 Comparison of APDCOL and EPDCOL: Problem 3

This problem is also Burgers' equation. We begin with the parameters $[t_0, t_{end}] = [0, 1]$ and $dt = 0.01$. Table 4.4 shows the L^2-norm error when $\epsilon = 10^{-3}$ and $TEPS = EPS = 10^{-4}$.

			NINT		
			20	30	40
KORD	4	EPDCOL	0.26	0.14	$8.05 \cdot 10^{-2}$
		APDCOL	$3.40 \cdot 10^{-4}$	$1.76 \cdot 10^{-4}$	$1.76 \cdot 10^{-4}$
	5	EPDCOL	0.19	0.13	$9.57 \cdot 10^{-2}$
		APDCOL	$9.05 \cdot 10^{-5}$	$8.11 \cdot 10^{-5}$	$5.41 \cdot 10^{-5}$
	6	EPDCOL	0.14	0.11	$6.36 \cdot 10^{-2}$
		APDCOL	$4.53 \cdot 10^{-5}$	$2.38 \cdot 10^{-5}$	$3.13 \cdot 10^{-5}$

Table 4.4: Problem 3: L^2-norm errors with $\epsilon = 10^{-3}$

From this table, one can observe that EPDCOL fails to solve that problem, while APDCOL still perform well. When $\epsilon = 10^{-4}$, the problem turns out to be more

difficult. Initially, we used the parameters $[t_0, t_{end}] = [0, 0.4]$, $dt = 0.01$ and $TEPS = EPS = 10^{-4}$. Table 4.5 gives the results.

			NINT					
			20		30		40	
			ERROR	NK	ERROR	NK	ERROR	NK
KORD	4	EPDCOL	0.40		0.38		0.36	
		APDCOL	$8.07 \cdot 10^{-4}$	40, 7	$5.57 \cdot 10^{-5}$	60, 7	$8.02 \cdot 10^{-5}$	40, 8
	5	EPDCOL	0.78		0.75		0.33	
		APDCOL	$3.44 \cdot 10^{-4}$	40, 7	$4.40 \cdot 10^{-4}$	60, 6	$1.28 \cdot 10^{-4}$	40, 8
	6	EPDCOL	0.33		0.25		0.29	
		APDCOL	$3.85 \cdot 10^{-4}$	40, 7	$1.07 \cdot 10^{-4}$	60, 8	$1.76 \cdot 10^{-4}$	40, 8

Table 4.5: Problem 3: L^2-norm errors when $\epsilon = 10^{-4}$ and $dt = 0.01$

Here NK means the final values of $NINT$ and $KORD$; the first number is $NINT$ and the second is $KORD$. From Table 4.5, we can see that APDCOL begins to struggle and achieves the error requirement by doubling $NINT$ and increasing $KORD$. We next change the parameter dt to 10^{-3}, the results are given in Table 4.6, only for the APDCOL code.

		NINT		
		20	30	40
KORD	4	$9.18 \cdot 10^{-4}$ ♣	$8.77 \cdot 10^{-4}$	$7.86 \cdot 10^{-4}$
	5	$7.57 \cdot 10^{-4}$	$5.70 \cdot 10^{-4}$	$3.75 \cdot 10^{-4}$
	6	$3.00 \cdot 10^{-4}$	$2.72 \cdot 10^{-4}$	$1.13 \cdot 10^{-4}$

Table 4.6: Problem 3: L^2-norm errors when $\epsilon = 10^{-4}$ and $dt = 10^{-3}$

Comparing Table 4.5 and Table 4.6, we conclude that dt is an important factor for our computational results. In a difficult problem, regions of rapid change may move a long distance in space over a short period of time. Therefore, if dt is chosen too large,

the wavefront may move a long distance during the interval $(T, T + dt)$ (see Figure 4.1, 4.2 for examples with large and small dt). Remeshing can then result in a mesh which is unsuitable for the profile at time T. But with a smaller dt, as in Figure 4.2, this will not happen.

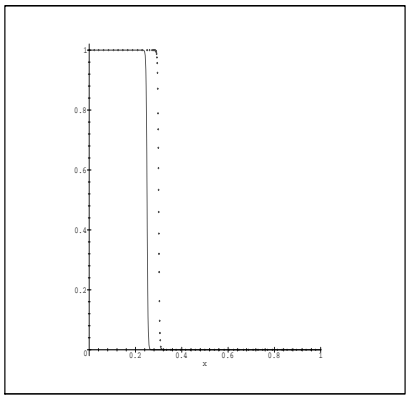

Figure 4.1: Example for a large dt

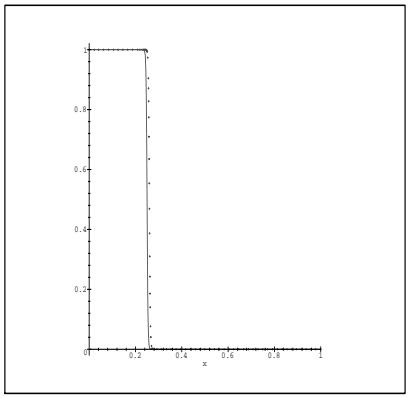

Figure 4.2: Example for a small dt

Here the continuous line shows the solution at T and the point shows the solution at $T + dt$. By comparing CPU time, it is also easy to see that using a smaller dt leads to much less CPU time than a larger one. For example, when $[t_0, t_{end}] = [0, 0.4]$,

$\epsilon = 10^{-4}$, $TEPS = EPS = 10^{-4}$, $NINT = 20$, $KORD = 5$ and $dt = 10^{-2}$, the CPU time is 198.509 seconds and it has performed in total 68 remeshings. But if we change dt to 10^{-3}, the CPU time is 49.2983 seconds and the total number of remeshings is 172. Therefore, future work will look at a way to evaluate the spatial error for every integrator step, and we will then not need the user supplied parameter dt.

From an examination of the previous tables, we note that different choices for $NINT$ and $KORD$ do not lead to significant different accuracy (The accuracy is reliable to the spatial and temporal tolerance). However, the costs associated with different choices of $NINT$ and $KORD$ are significantly different. Using the same parameters as in Table 4.6, CPU time and total remeshing times are shown in Table 4.7.

		NINT					
		20		30		40	
		CPU	TRT	CPU	TRT	CPU	TRT
KORD	4	49.7554 ♣	115	42.4466	148	49.2157	112
	5	49.2983	172	55.0256	98	65.4356	75
	6	60.3446	116	72.5928	72	89.6984	63

Table 4.7: Problem 3: CPU time and remeshing times when $\epsilon = 10^{-4}$ and $dt = 10^{-3}$

TRT means total remeshing times. It seems that for difficult problems, small $NINT$ and $KORD$ with frequent remeshing is a better choice than large $NINT$ or $KORD$. Therefore, one goal of our future work is to find the optimal $NINT$ and $KORD$ which just satisfy the tolerance and do not lead a doubling of $NINT$ or an increase in $KORD$.

As we know, there are two significant parts to the CPU time. The first part is the remeshing time which includes the time for calculating the error, the time for

remeshing and the time for calculating the initial condition for restarting. The second part is the integration time between two remeshings. If $NINT$ and $KORD$ are larger, the code will need fewer remeshings. Therefore, the first part will be less. However, the interval between two remeshings will be longer, and during that time interval the mesh may become relatively poorly distributed. This will imply that the ODE solver will take smaller and smaller step size to meet the required tolerance, and the second part of the cost can be extremely large. This leads to a dilemma. Since we use a high-order method, we do not want to remesh too frequently. On the other hand, if we use fewer remeshings, the time for ODE solver can be quite large. It may be that the ODE solver should stop and remesh even if the error is met. This difficuty may be solved by EP. By giving a bound for how closely the mesh should be equidistribed, we can force more frequent remeshings.

We should emphasize that APDCOL is an experimental code. During our experiments, some other error measures were considered. For example, the first one we considered was the L^2-norm error. That is, we tried to have

$$\sqrt{\int_{x_i}^{x_{i+1}} (ui - uii)^2 dx} = \text{constant}. \qquad \text{(MON1)}$$

We call this error measure as MON1. Similarly, we have tried MON2, MON3 as

$$\sqrt[4]{\int_{x_i}^{x_{i+1}} (ui - uii)^2 dx} = \text{constant}. \qquad \text{(MON2)}$$

$$\sqrt[8]{\int_{x_i}^{x_{i+1}} (ui - uii)^2 dx} = \text{constant}. \qquad \text{(MON3)}$$

To compare those error measures within our code, we first tried an easy problem obtained by using

$$\epsilon = 10^{-3},$$

$$[t_0, t_{end}] = [0, 0.4], \qquad dt = 10^{-3},$$

$$TEPS = EPS = 10^{-4},$$

and

$$NINT = 20, \qquad KORD = 4.$$

We observed that MON1 caused to code to struggle and we can not reach the tolerance even though $NINT$ is increased to 80 and $KORD$ is increased to 10. MON2 leads to the code doubling $NINT$ in the end. We next changed dt to 10^{-2}. From Table 4.4, we can see that it is not a difficult problem for APDCOL. This time MON2 causes the code to struggle and we can not reach the tolerance even though $NINT$ is changed to 80 and $KORD$ is change to 10. To compare MON3 with APDCOL, a very difficult problem (see Table 4.5) is chosen with

$$\epsilon = 10^{-4}, \qquad dt = 10^{-2},$$

$$[t_0, t_{end}] = [0, 0.4], \qquad TEPS = EPS = 10^{-4},$$

$$NINT = 20, \qquad KORD = 5.$$

Using APDCOL, $NINT = 40$, $KORD = 7$, the CPU time is 198.509 seconds and total remeshing times is 68; while by using APDCOL with MON3, $NINT = 80$, $KORD = 9$, the CPU time is 504.444 seconds and total remeshing times is 41. We can see that MON3 cause APDCOL to require larger $NINT$ and $KORD$ values. This supports the claim that after remeshing, APDCOL can give finer grids than APDCOL using MON3.

Chapter 5

Conclusion and Future Work

In this thesis, an adaptive algorithm was presented, based on the Equidistribution Principle and the conventional method of lines. The L^2-norm error estimate was used to control mesh refinement and changes in the order of the polynomial approximation. For $NPDE = 1$, we implemented our algorithm in the experimental Fortran 77 code, APDCOL, which is described in Chapter 3. In Chapter 1, we introduced the traditional method of lines briefly, and a review of some recent approaches for adaptive methods was presented in Chapter 2.

The results presented in Chapter 4 confirm that APDCOL is able to solve a class of parabolic problems whose solutions have regions of rapid change. It performed well on all the problem we have tested. Computational results support the assumption that our error estimate converges to the true error in the L^2-norm as $h \to 0$. A comparison of L^2-norm errors between APDCOL and EPDCOL showed that APDCOL is superior to EPDCOL when dealing with these difficult problems. However, since APDCOL is an experimental code, there is a lot left for the future work. A summary for future work in this area follows.

- Use a hot start after remeshing.

- Find a method to evaluate the spatial error at the end of each time step. If this can be done, we will not need the user supplied parameter dt any more.
- Find a way to determine optimal values of $NINT$ and $KORD$, i.e., construct an hp-version.
- The new code should be more dependent on the equidistribution principle. That is, even if the L^2-norm error requirement is met after a time step, wherever the error estimate is not equidistributed, we should remesh.
- The package should handle a system of PDEs.

Appendix A

Test Problems

These problems have been used by many authors, and the exact solution is known for each problem; this is helpful for the evaluation of the software. Some problems include one or more parameters which can be varied to adjust the difficulty of the problem. We now present our test problems.

A.1 Problem 1

Our first problem is an easy one; it is the second example of [MS79].

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

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

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

The exact solution for this problem is easily seen to be

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

See Figure A.1.

A.2 Problem 2

This is the Burgers' equation:

$$u_t = -uu_x + \epsilon u_{xx}, \qquad 0 < x < 1, \quad t > 0, \tag{A.1a}$$

$$u(x,0) = u_0(x), \qquad 0 \le x \le 1, \tag{A.1b}$$

$$u(0,t) = b_L(t), \qquad u(1,t) = b_R(t), \qquad t \ge 0, \tag{A.1c}$$

where the viscosity parameter, ϵ, is chosen as 3×10^{-3} or 10^{-3}. Here the initial conditions and the boundary conditions are decided by the exact solutions.

This problem is taken from [BSV88, AAF95], and has the exact solution

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

and

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

See Figure A.2, A.3.

A.3 Problem 3

Our third problem is also Burgers' equation. The exact solution is

$$u(x,t) = c - d\tanh\{\frac{d}{2\epsilon}(x - ct - x_0)\},$$

where $c = \frac{1}{2}(u_\infty^- + u_\infty^+)$, $d = \frac{1}{2}(u_\infty^- - u_\infty^+)$, $u_\infty^- - u_\infty^+ > 0$, where u_∞^- and u_∞^+ are parameters. It represents a travelling wave with velocity c and initial location controled by x_0. Following [BSV88] we used the parameter values $u_\infty^- = 1$, $u_\infty^+ = 0$ and $x_0 = \frac{1}{4}$. In our numerical experiments, we choose ϵ as 10^{-3} or 10^{-4}.

See Figure A.4, A.5.

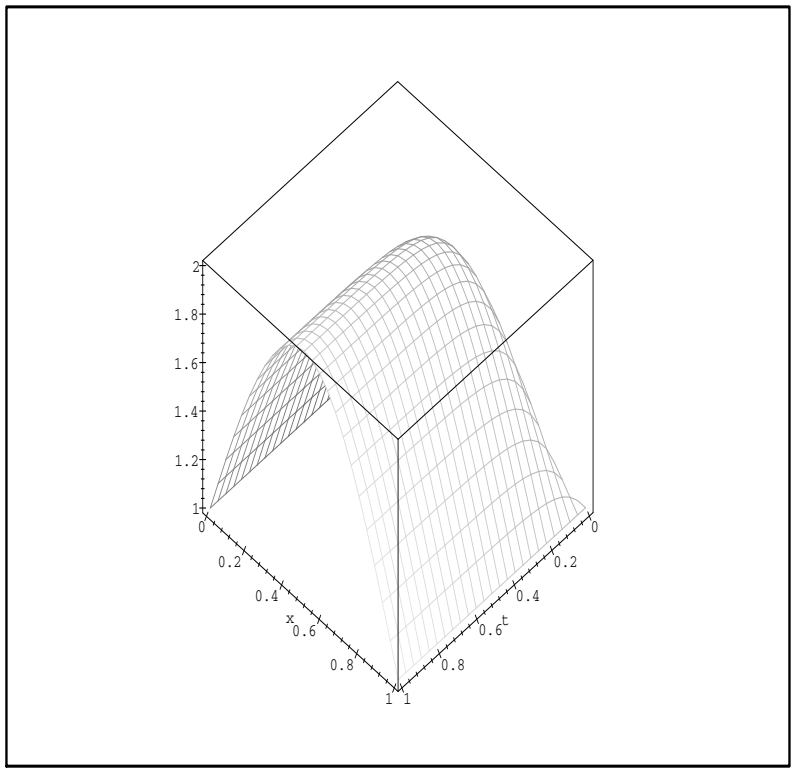

Figure A.1: Problem 1

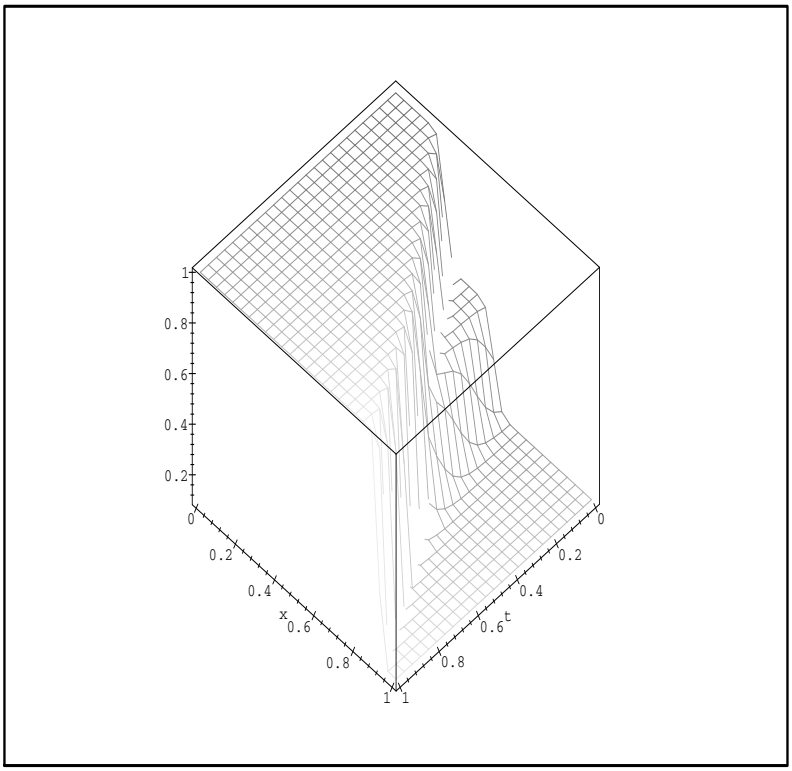

Figure A.2: Problem 2 with $\epsilon = 3 \times 10^{-3}$

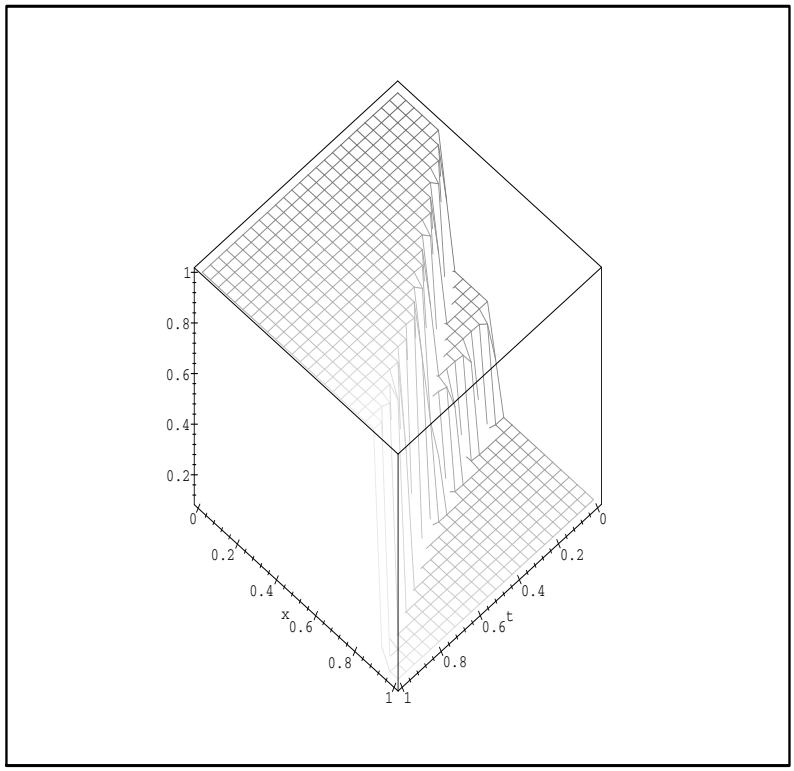

Figure A.3: Problem 2 with $\epsilon = 10^{-3}$

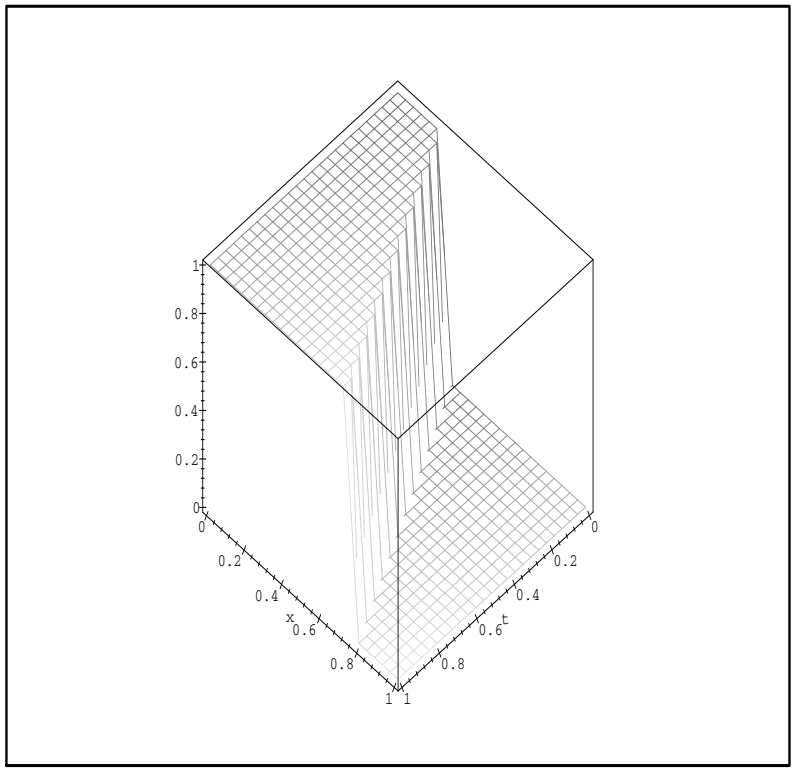

Figure A.4: Problem 3 with $\epsilon = 10^{-3}$

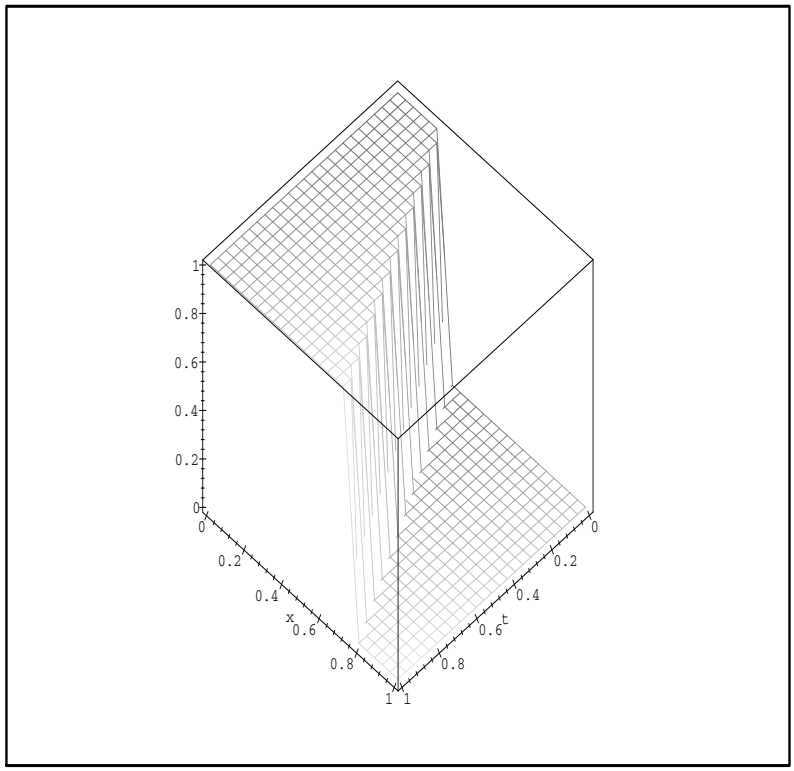

Figure A.5: Problem 3 with $\epsilon = 10^{-4}$

Bibliography

[AAF95] 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 (1995), pp. 520-543.

[AF86a] 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 (1986), pp. 778-796.

[AF86b] 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 (1986), pp. 3-26.

[AFMW92] S. Adjerid, J.E. Flaherty, P.K. Moore and Y.J. Wang, *High-Order Adaptive Methods for Parabolic Systems*, Physica D, 60 (1992), pp. 94-111.

[AFW93] S. Adjerid, J.E. Flaherty and Y.J. Wang, *A Posteriori Error Estimation with Finite Element Methods of Lines for One-Dimensional Parabolic Systems*, Numer. Math., 65 (1993), pp. 1-21.

[B73] C.De Boor, *Good Approximation by Splines with Variable Knots II*, in Springer Lecture Notes Series 363, Springer-Verlag, Berlin, 1973.

[B77] C.De Boor, *Package for Calculating with B-Splines*, SIAM J. Numer. Anal., 14 (1977), pp. 441-472.

[B87] J.C. Butcher, *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and Gereral Linear Methods*, 1987.

[BB82a] M. Bieterman and I. Babuska, *The Finite Element Method for Parabolic Equations. I. A Posteriori Error Estimation*, Numer. Math., 40 (1982), pp. 339-371.

[BB82b] M. Bieterman and I. Babuska, *The Finite Element Method for Parabolic Equations. II. A Posteriori Error Estimation and Adaptive Approach*, Numer. Math., 40 (1982), pp. 373-406.

[BCHR95] C.J. Budd, J. Chen, W. Huang and R.D. Russell, *Moving Mesh Methods with Applications to Blow-up Problems for PDEs*, Numerical Analysis 1995: Proc. of 1995 Biennial Conference on Numerical Analysis, edited by D.F. Griffiths and G.A. Watson, Pitman Research Notes in Mathematics, Longman Scientific and Technical (1996), pp. 1-17.

[BDF89] 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 (1989), pp. 375-397.

[BHK85] R.F. Boisvert, S.E. Howe and D.K. Kahaner, *GAMS: A Framework for the Management of Scientific Software*, ACM Trans. Math. Softw., 11 (1985), pp. 313-355.

[BHR96] C.J. Budd, W. Huang and R.D. Russell, *Moving Mesh Methods for Problems with Blow-up*, SIAM J. Sci. Comput., 17 (1996), pp. 305-327.

[BS73] C.De. Boor and B. Swartz, *Collocation at Gaussian Points*, SIAM J. Numer. Anal., 10 (1973), pp. 582-606.

[BSV88] 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 (1988), pp. 191-213.

[CFL86] J.M. Coyle, J.E. Flaherty and R. Ludwig, *On The Stability of Mesh Equidistribution Strategies for Time-Dependent Partial Differential Equations*, J. Comput. Phys., 62 (1986), pp. 26-39.

[D79] J.C. Diaz, *Collocation-H^{-1}-Galerkin Method for Parabolic Problems with Time Dependent Coefficients*, SIAM J. Numer. Anal., 16 (1979), pp. 911-922.

[DD87] E.A. Dorfi and L.O'C. Drury, *Simple Adaptive Grids for 1-D Initial Value Problems*, J. Comput. Phys., 69 (1987), pp. 175-195.

[DFK83] 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, 3 (1983), pp. 358-375.

[F78] G. Fairweather, *Finite Element Galerkin Methods for Differential Equations*, Lecture Notes in Pure and Applied Mathematics, 1978

[FVZ90] R.M. Furzeland, J.G. Verwer and P.A. Zegeling, *A Numerical Study of Three Moving-Grid Methods for One-dimensional Partial Differential Equations Which Are Based on The Methods of Lines*, J. Comput. Phys., 89 (1990), pp. 394-388.

[G71] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, 1971.

[GB86a] W. Gui and I. Babuska, *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 (1986), pp. 577-612.

[GB86b] W. Gui and I. Babuska, *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 (1986), pp. 613-657.

[GB86c] W. Gui and I. Babuska, *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 (1986), pp. 659-683.

[H83] A.C. Hindmarsh, *ODEPACK: A Systematized Collection of ODE Solvers*, in R.S. Stepleman et. al. eds., Scientific Computations (1983) pp. 55-64.

[HGH91] D.F. Hawken, J.J. Gottlieb and J.S. Hansen, *Review of Some Adaptive Node-Movement Techniques in Finite-Element and Finite-Difference Solutions of Partial Differential Equations*, J. Comput. Phys., 95 (1991), pp. 254-302.

[HR96] W. Huang and R.D. Russell, *A Moving Collocation Method for Solving Time Dependent Partial Differential Equations*, Appl. Numer. Math., 20 (1996), pp. 101-116.

[HR97] W. Huang and R.D. Russell, *Analysis of Moving Mesh Partial Differential Equations with Spatial Smoothing*, SIAM J. Numer. Anal., 34 (1997), pp. 1106-1126.

[HR99] W. Huang and R.D. Russell, *MOVCOL: Moving Collocation Software for Time Dependent PDE*, unpublished software, 1999.

[HRR94a] W. Huang, Y. Ren and R.D. Russell, *Moving Mesh Partial Differential Equations (MMPDEs) Based on the Equidistribution Principle*, SIAM J. Numer. Anal., 31 (1994), pp. 709-730.

[HRR94b] W. Huang, Y. Ren and R.D. Russell, *Moving Mesh Methods Based on Moving Mesh Partial Differential Equations*, J. Comput. Phys., 113 (1994), pp. 279-290.

[KM91] P. Keast and P.H. Muir, *Algorithm 688. EPDCOL: A more Efficient PDECOL Code*, ACM Trans. Math. Softw. 17,2 (1991), pp. 153-166.

[KMN96] 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*, Numerical Analysis, World Scientific Publishing, (1996), pp. 207-224.

[LBD91] 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 (1991), pp. 573-594.

[M81] K. Miller, *Moving Finite Elements. II*, SIAM J. Numer. Anal., 18 (1981), pp. 1033-1057.

[MF90] P.K. Moore and J.E. Flaherty, *A Local Refinement Finite-element Method for One-dimensional Parabolic Systems*, SIAM J. Numer. Anal., 27 (1990), pp. 1422-1444.

[MM81] K. Miller and R. N. Miller, *Moving Finite Elements. I*, SIAM J. Numer. Anal., 18 (1981), pp. 1019-1032.

[MS79] N.K. Madsen and R.F. Sincovec, *Algorithm 540. PDECOL, General Collocation Software for Partial Differential Equations*, ACM Trans. Math. Softw. 5,3 (1979), 326-351.

[N95] T.B. Nokonechny, *The Method of Lines Using Monomial Spline Collocation for Parabolic Partial Differential Equations*, M.Sc thesis, Math. Dept., Dalhousie University, 1995.

[P82] L.R. Petzold, *A Description of DASSL: A Differential/algebraic System Solver*, SAND82-8637, Sandia Labs, Livermore, CA (1982).

[P89] L.R. Petzold, *An Adaptive Moving Grid Method for One-dimensional Systems of PDEs and its Numerical Solution*, in Adaptive Methods for Partial Differential Equations (SIAM, Philadelphia, PA, 1989), pp. 253-265. Edited by J.E. Flaherty, P.J. Paslow, M.S. Shephard and J.D. Vasilakis.

[SC86] J.M. Sanz-Serna and I. Christie, *A Simple Adaptive Technique for Nonlinear Wave Problems*, J. Comput. Phys., 67 (1986), pp. 348-360.

[SK77] R.D. Skeel and A. Kong, *Blended Linear Multistep Methods*, ACM Trans. Math. Softw. 3 (1977), pp. 326-345.

[SW76] L.F. Shampine and H.A. Watts, *Globe Error Estimation for Ordinary Differential Equations*, ACM Trans. Math. Softw. 2 (1976), pp. 172-186.

[T84] V. Thomée, *Galerkin Finite Element Methods for Parabolic Problems*, volume 1054 of Lecture Notes in Mathematics, 1984.

[VBFZ89] J.G. Verwer, J.G. Blom, R.M. Furzeland and P.A. Zegeling, *A Moving Grid Method for One-dimensional PDEs Based on the Method of Lines*,in Adaptive Methods for Partial Differential Equations (SIAM, Philadelphia, PA, 1989), pp. 160-175. Edited by J.E. Flaherty, P.J. Paslow, M.S. Shephard and J.D. Vasilakis.

[VBS89] J.G. Verwer, J.G. Blom and J.M. Sanz-Serna, *An adaptive Moving Grid Method for One-dimensional Systems of Partial Differential Equations*, J. Comput. Phys., 82 (1989), pp. 454-486.

[W75] M.F. Wheeler, *An H^{-1} Galerkin Method for Parabolic Problems in a Single Space Variable*, SIAM J. Numer. Anal., 12 (1975), pp. 803-817.

[W79] A.B. White, *On Selection of Equidistributing Meshes for Two-point Boundary-value Problems*, SIAM J. Numer. Anal., 16 (1979), pp. 472-502.

[W82] A.B. White, *On the Numerical Solution of Initial/Boundary-value Problems in One Space Dimension*, SIAM J. Numer. Anal., 19 (1982), pp. 683-697.