Exploration of Moving Transformation Methods for Boundary Value Ordinary Differential Equations and One-Dimensional Time-Dependent Partial Differential Equations

By

Connor Tannahill

A Thesis Presented to

Saint Mary's University, Halifax, Nova Scotia

in Partial Fulfillment of the Requirements for

the Degree of Bachelor of Science (Honours).

April 26, 2019, Halifax, Nova Scotia

Approved: Dr. Paul Muir

Supervisor

Approved: Dr. Walt Finden

Reader

Date: April 26, 2019

Abstract

Rapid advances in computing power have given computational analysis and simulation a prominent role in modern scientific exploration. Differential equations are often used to model complex scientific phenomena. In practice, these equations can not be solved exactly and numerical approximations which accurately preserve the characteristics of the modelled phenomena must be employed. This has motivated the development of accurate and efficient numerical methods and software for these problems. This thesis explores a class of adaptive methods for accurately computing numerical solutions for two common classes of differential equations, boundary value ordinary differential equations and time-dependent partial differential equations in one spatial dimension. These adaptive methods, referred to as moving transformation (MT) methods, are used to improve the accuracy of standard numerical methods for these problem classes and can be extended to higher dimensions. MT methods improve the accuracy of these standard numerical algorithms by transforming the differential equation into a related differential equation on a computational domain where it is easier to solve. The solution to this transformed differential equation can then be transformed back to the original physical domain to obtain a solution to the original differential equation. Software implementing MT methods is developed and computational experiments performed to determine the effectiveness of these methods compared to traditional adaptation approaches. We also investigate the suitability of these methods for implementation in adaptive error control algorithms.

Contents

Acknowledgments 1

Introduction **2**

1 Background **8**

1.1 Numerical Methods for BVODEs 8

1.2 Overview of *bvode* / COLNEW 12

1.3 Numerical Methods for PDEs 14

1.4 Overview of BACOLI 18

1.5 Adaptation Strategies and the Moving Transformation Method 21

2 Moving Transformation Methods for BVODEs **34**

2.1 Computing the Coordinate Transformation 38

2.2 Moving Transformation Approach for BVODEs Using an Arc-Length Monitor Function 52

2.3 Moving Transformation Approach for BVODEs Using an Error Estimate Monitor Function 61

2.4 Adaptive Error Control Strategy for BVODEs Using MT Methods 67

3 Moving Transformation Methods for PDEs **73**

3.1 Computing the Coordinate Transformation 78

3.2 Moving Transformation Approach for PDEs Using an Arc-Length Monitor Function 88

3.3 Moving Transformation Approach for PDEs Using an Error Estimate Based Monitor Function 96

3.4 Outline of an Adaptive Error Control Algorithm using Moving Transformation Methods100

3.5 Moving Transformation Approaches in 2D 106

4 Conclusions & Future Work 110

4.1 Conclusions 110

4.2 Future Work 111

References . . . .

List of Figures

1 PDE solution on the physical domain Ω_p and mapped on the computational domain Ω_c. 6

1.1 Midpoint scheme applied with different numbers of mesh subintervals. 11

1.2 Finite Difference Method of Lines algorithm, applied with increasing N values. Plotted against the exact solution. 17

1.3 $f(x) = 5x + 1$ on uniform (top) and equidistributed (bottom) meshes. 23

1.4 Quasi-Lagrange and rezoning MT approaches. 31

1.5 Relationship between Ω_c and Ω_p. 32

2.1 Solutions, $y_1(x)$, to BVODE (2.3), for several λ values. 37

2.2 Arc-length monitor function for BVODE (2.3) for several choices of the λ parameter. 42

2.3 Smoothing of the exact arc-length monitor function for (2.3) with $\lambda = 50$. 42

2.4 In the top plot we show the coordinate transformation $x(\xi)$ obtained from the numerical solution to MMODE0 (2.14) along with a normalized plot of the exact arc-length monitor function for (2.3) with $\lambda = 10$. In the bottom plot we show the mesh obtained from de Boor's algorithm based on the same monitor function as in the top plot. . . 46

2.5 Per-subinterval departure from equidistribution for MMODE0 generated mesh (blue) and uniform mesh (red) in Figure 2.4. Plotted against the target per-subinterval value, shown in green. 47

2.6 Comparison of smoothed MMODE0 solution $x(\xi)$ with the de Boor generated mesh of $N = 20$ subintervals. $x(\xi)$ is based on the interpolated exact arc-length monitor function for (2.3) with $\lambda = 50$, with 5 iterations of (2.6) applied; $\bar{x}$ was obtained using the exact arc-length monitor function. $E_{eq}(x(\xi)) = 1.31$ and $E_{eq}(\bar{x}) = 0.54$. 48

2.7 Solution to (2.3) with $\lambda = 100$, smoothed using the arc-length monitor function in the MT method with the Midpoint scheme used for spatial discretization, $N = 20$. $\hat{y}(x(\xi))$ is the transformed computed solution on $\Omega_c = [0, 1]$. $y(x(\xi))$ is the transformed computed solution on $\Omega_p = [0, 1]$. 56

2.8 Solution to (2.3) with $\lambda = 50$ computed using the MT algorithm and then transformed onto Ω_p. Plotted against solution computed directly on a uniform mesh on Ω_p; $N = 20$. 56

2.9 Solution to (2.3) with $\lambda = 100$ computed using the MT algorithm and then transformed onto Ω_p. Plotted against solution computed directly on a uniform mesh on Ω_p; $N = 40$. 57

2.10 Solution to (2.3) with $\lambda = 500$ computed using the MT algorithm and then transformed onto Ω_p. Plotted against solution computed directly on a uniform mesh on Ω_p; $N = 80$. 57

2.11 Comparison of the monitor function (2.19) with $E(Y(x))$ the exact error against the exact error, for an approximate solution to (2.3) with $\lambda = 100$, discretized with the midpoint scheme on a uniform mesh of 40 subintervals on Ω_p. 63

2.12 Comparison of the monitor function (2.24) with the exact error for a solution to (2.3) with $\lambda = 100$, discretized using the Midpoint scheme on a uniform mesh of 40 subintervals on Ω_p. 65

3.1 Solution to the One Layer Burgers' Equation, $\epsilon = 10^{-3}$. 76

3.2 Solution to the Two Layer Burgers' Equation, $\epsilon = 10^{-3}$. 77

3.3 Coordinate transformation and normalized arclength monitor function at several points in time for OLBEϵ3, $t = 0$ (top), $t = \frac{1}{2}$ (middle), $t = 1$ (bottom). 5 iterations of monitor function smoothing, $\tau = 10^{-2}$. The spike in $M(x,t)$ corresponds to the location of the travelling layer region in the solution to OLBEϵ3. 87

3.4 OLBEϵ3. Solved using finite difference discretization (1.13) using a MT method with arc-length monitor function, $\tau = 10^{-2}$, $N = 20$ subintervals, and 5 iterations of monitor function smoothing. 93

3.5 OLBEϵ3. Solved directly using finite difference discretization (1.13) on a uniform mesh with $N = 20$ subintervals. 94

3.6 OLBEϵ3. Exact Solution 94

3.7 OLBEϵ2. Solved using finite difference discretization (1.7)-(1.9) with $N = 40$ using a MT method with error estimation monitor function. 99

3.8 TLBEϵ2. Solved using finite difference discretization (1.7)-(1.9) with $N = 40$ using a MT method with error estimation monitor function. 99

List of Tables

2.1 E_{eq} (2.3) with for varying λ choices and number of iterations of smoothing scheme (2.6). E_{eq} is computed with $N = 20$. 49

2.2 CPU time when solving MMODE0 for different λ choices and number of smoothing iterations with *bvode*. 50

2.3 E_{eq} values from iterative generation of the coordinate transformation, 5 iterations of monitor function smoothing, $N = 40$. 51

2.4 Effect of *bvode* error tolerance choice on the quality on E_{eq}, $\lambda = 100$, $N = 20$. 52

2.5 Error of the numerical solution computed by the MT method using the arc-length monitor function with various degrees of monitor function smoothing and spatial discretization is done using the Midpoint scheme. Applied to (2.3) for several λ values and several values for N. 59

2.6 Error of the numerical solution obtained through the direct discretization of (2.3) on Ω_p using Midpoint scheme on a non-uniform mesh generated by evaluating the solution to MMODE0 at the mesh points of a uniform mesh of N subintervals on Ω_c. The arc-length monitor function was used with various degress of monitor function smoothing for use in MMODE0. Applied to (2.3) for several λ values and several N values. 60

2.7 Error results for the MT algorithm using the error estimate monitor function (2.24), applied to (2.3) with several values of λ, N, and numbers of monitor function smoothing iterations. 66

2.8 Error results for direct discretization on Ω_p using the non-uniform mesh generated through the evaluation of MMODE0 at the mesh points of a uniform mesh of N subintervals on Ω_c, where the error estimation monitor function was used in MMODE0. 67

2.9 Errors after successive mesh doubling for MT method using fixed coordinate transformation generated using the error estimate monitor function. 70

2.10 Exact error for solutions to (2.3) with several λ values and a simple error control algorithm which requires that the error, as estimated by (2.25), satisfy various tolerance values. 72

3.1 E_{eq} for varying problems and parameter choices for solving MMPDE5 with BACOLI. E_{eq} was computed using $N = 160$. 83

3.2 CPU time for varying parameter choices when solving MMPDE5 with BACOLI. . . 84

3.3 Error results for arc-length monitor function MT method, $\tau = 10^{-2}$. 95

3.4 Error results for arc-length monitor function MT method, $\tau = 10^{-1}$. 95

Acknowledgments

I would first like to thank my supervisor Dr. Paul Muir. His rigorous approach to science, computing, and scientific computing is something that I will take with me wherever I go. His perspectives on technology, communication, and technical communication have shaped the ways I view the work I do and how to strive for mutual understanding of my work with both the public and other research domains.

Thank you to my reader Dr. Walt Finden for this close reading of my thesis and his helpful critiques.

Thank you to my family for their constant support throughout my undergraduate studies.

Special thanks to my friends and fellow graduating honours students Matthew Rafuse, Will Scherer and Jordan Dempsey. Additional thanks to my lab-mate Owen Sharpe.

Introduction

Differential equations are fundamental tools in the mathematical modelling of complex phenomena and have been applied in many diverse application areas such as image processing [1], epidemiology [2], and weather prediction [3]. Generally, differential equations cannot be solved exactly; this is a fundamental difficulty which has led to the wide application of numerical methods to generate approximate solutions to these problems. Sophisticated algorithms and software have been developed which efficiently compute accurate numerical solutions for general classes of differential equations. High-quality numerical software packages typically implement adaptive error control algorithms in order to generate solutions for which an estimated error is less than a user-provided error tolerance. When solving differential equations, it is vital that the numerical solution preserves the important physical characteristics of the system being modelled. By bounding the estimated error of the solution to a within a tolerance, error control solvers for differential equations provide assurance that a solution is computed which is as accurate as is required for a given application and hence the solution will be sufficiently representative of the system being modelled. The adaptation approaches used in these solvers can also provide a level of computational efficiency since the algorithm need only do as much work as is required to attain the user tolerance. High quality, numerical library level software efficiently implements its component numerical algorithms and undergoes extensive performance analysis and testing to ensure efficiency and robustness. With these tools, a user can generate solutions which are sufficiently accurate for the given application, without having to make simplifications to their model.

In this thesis, we consider adaptive methods for the numerical solution of differential equations which can be applied to two common classes of differential equations. The first class is Boundary Value Ordinary Differential Equations (BVODEs) having the general form

$$\underline{y}'(x) = \underline{f}(x, \underline{y}(x)), \quad x_a \leq x \leq x_b, \tag{1}$$

with separated boundary conditions (BCs)

$$\underline{g}(\underline{y}(x_a), \underline{y}(x_b)) = \underline{0}. \tag{2}$$

The second class of equations considered are One Dimensional Partial Differential Equations (PDEs) of the form

$$\begin{gathered} \underline{u}_t(x,t) = \underline{f}(x, t, \underline{u}(x,t), \underline{u}_x(x,t), \underline{u}_{xx}(x,t)), \\ x_a \leq x \leq x_b, \quad t_0 \leq t \leq t_{out}, \end{gathered} \tag{3}$$

having the separated BCs

$$\begin{gathered} \underline{b}_L(t, \underline{u}(x_a,t), \underline{u}_x(x_a,t)) = \underline{0}, \\ \underline{b}_R(t, \underline{u}(x_b,t), \underline{u}_x(x_b,t)) = \underline{0}, \end{gathered} \tag{4}$$

and initial conditions,

$$\underline{u}(x, t_0) = \underline{u}_0(x). \tag{5}$$

These classes of equations have been frequently employed in mathematical models occurring in many application domains. The spatial components of each of these problem classes motivates important similarities in the numerical methods and adaptation approaches used to solve them.

Adaptive numerical methods are those which adjust the way the computation is performed in response to the relative difficulty of certain components of the problem. Adaptation may be based on either *a priori* or *a posteriori* considerations. In the first case, some information about the solution behaviour is known prior to computing it. In the case of BVODE and PDE problems, this may be informed by factors such as the expected behaviour of the system being modelled, or known error bounds of the numerical method being used to solve the given equation [4, 5]. Alternatively, adaptation based on *a posteriori* information typically requires that some initial approximate solution be computed, with no expectations about its accuracy. Properties of this initial approximation are then measured to determine if and where the solution must be improved. In the problem classes considered here, typical *a posteriori* adaptation methods make use of measures of the initial solution arclength, curvature or its error as indicated by an associated error estimate [5].

The adaptive methods considered in this thesis can make use of either *a priori* or *a posteriori* knowledge and therefore can be effectively applied in many contexts. For BVODEs and PDEs, the difficulty in computing accurate numerical solutions using standard algorithms is typically due to regions of rapid change in the solution in the spatial domain. Adaptive methods compensate for this difficulty by locally refining the computation in regions of large solution variation so as to compensate for the large solution error in these regions [4, 5]. Alternatively, a global refinement strategy can be used, where the order of the numerical method is increased such that the entire solution is of higher accuracy, though this is less common [5].

The methods which are the primary focus in this thesis are referred to in the literature as Moving Mesh (MM) methods, but here will be referred to as Moving Transformation (MT) methods. We use this alternative name to distinguish these methods from many of the standard adaptation approaches for these problems, which are also commonly referred to as MM methods. MT methods take an alternative but related view of adaptivity from many standard approaches. A continuous, invertible mapping $x : \Omega_c \mapsto \Omega_p$ between a computational domain, Ω_c, and a physical domain, Ω_p [5, 4]. In the context of an MM method, the purpose this transformation is to generate a set of discrete points at which the differential equation is evaluated to facilitate a more accurate numerical approximation of the differential equation. When used in an MT method, the transformation is used to map a physical differential equation on Ω_p to a transformed differential equation on Ω_c such that regions of large solution variation in Ω_p are smoothed on Ω_c [5]. The coordinate transformation stretches the independent variable in Ω_c so that regions of Ω_p where the solution of the physical differential equation vary rapidly correspond to regions in Ω_c where the solution to the transformed differential equation in Ω_c has less variation. Figure 1 shows an example of the solution to a PDE at a fixed point in time, plotted both after it has been transformed on to the computational domain as well as on the original physical domain. This figure demonstrates the effect of this smoothing of the solution, as the steep region in Ω_p becomes substantially smoothed in Ω_c. The motivation behind this procedure is that the transformed equation on Ω_c can be approximated more accurately and efficiently than the original. Once a solution to the transformed equation is computed, it is then mapped back to the physical domain Ω_p, giving a solution to the original equation.

MT methods can be thought of as an indirect approach to adaptation. This is because instead of directly addressing the difficulties of a given problem, the problem itself is instead changed to make it more easily solvable. In the more common indirect methods, the adaptation is not performed on the problem itself, but within the numerical methods used to solve it. There are many solvers which implement direct methods for both of the problem classes considered in this thesis, with COLNEW [6] and BACOLI [7] packages being prominent examples of error control solvers for BVODEs and PDEs, respectively. Due to the complexity, relatively poor efficiency for 1D problems, and the fact that the methods are relatively new, few general-purpose software packages for these problem classes which implement MT methods have been developed. To our knowledge, no general-purpose MT solvers for BVODEs have been developed. MOVCOL [8] is a PDE solver implementing high order numerical methods that yield continuous solution approximations; however, MOVCOL does not implement MT methods in a strict sense, as some components of MT methods are used to simply generate an adapted spatial mesh which is used to solve the physical equation directly on Ω_p.

For BVODEs and PDEs in one spatial dimension, direct refinement approaches are generally more efficient than MT methods. However, in the case of multidimensional PDEs, the MT approach provides a powerful framework for efficient adaptation, a substantial issue when solving these equations [4]. The computational advantages provided by this method will be described in more detail in Chapters 2 and 3.

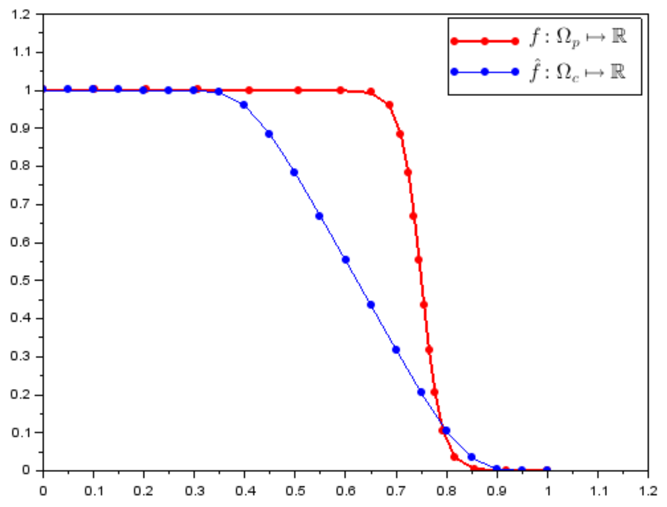


Figure 1: PDE solution on the physical domain Ω_p and mapped on the computational domain Ω_c.

MT methods have been studied in the literature for a number of years from a primarily mathematical perspective in order to resolve the various theoretical difficulties of the approach. From this research, MT methods have reached a point where robust algorithms using the MT approach can be developed for 1D problems, with substantial progress having been made in the multi-dimensional case [5]. Despite this, there have been few attempts at a high-quality software implementation for general problem classes. The software which does exist is generally application specific and lacks many standard features common in high-quality numerical software packages for differential equations. For example, to our knowledge, there are no MT solvers that implement adaptive spatial error control, and further, most software that does exist generate only low-order, discrete solution approximations rather than the high-order continuous approximations that are typically generated by standard high-quality differential equation solvers. Continuous solution approximations not only provide a convenience for the user but have many algorithmic benefits for adaptive solvers, such as for use in error estimation as well as propagation of previously obtained solution information following adaptation. It is therefore the goal of this thesis to explore algorithms based on MT methods which can be extended to provide high-quality implementations of these methods. This involves the exploration of MT methods for adapting the computation of solutions to BVODE and PDE problems, eventually leading to a discussion of adaptive error control algorithms for each of these problem classes. The approaches considered here could be used to inform the development of 1D PDE error control software, and generalizations of these approaches could be applied to PDEs in two spatial dimensions (2D PDEs), where the advantages of using the MT approach are more clear.

An example of a software project where MT methods may be useful is BACOL2D [9], a 2D PDE solver which implements a tensor product B-spline Gaussian collocation algorithm for spatial discretization on rectangular 2D grids. MT methods may be an effective approach for providing error control for this type of algorithm since adaptation can be performed without having to use a non-rectangular grid.

This thesis is organized as follows. Chapter 1 gives relevant background materials and review of the literature. Chapter 2 goes into the exploration and development of BVODE MT algorithms. Chapter 3 extends these approaches to the 1D PDE case. Chapter 4 finishes with conclusions on the results of this research and provides some suggestions for future work.

Chapter 1

Background

1.1 Numerical Methods for BVODEs

Numerical methods for BVODEs have been studied and applied extensively. Initial attempts at developing numerical methods for BVODEs were based on the interpretation of the problems as a special case of Initial Value Ordinary Differential Equation (IVODE) problems, which are problems for which the exact solution is known only at some initial point in time. To translate a BVODE problem into an IVODE problem, the available left-hand side boundary conditions are taken as initial conditions, and the remaining missing initial conditions are guessed at. This led to the development of numerical methods, known as shooting methods, which could take advantage of pre-existing solvers for IVODEs [10]. With the shooting method, the right-hand side boundary condition is taken to be a constraint which the solution must satisfy, and the IVODE is repeatedly solved with a Newton-type iteration until a solution satisfying the right-hand side boundary conditions is generated, yielding an approximate solution to the BVODE [10]. However, algorithms such as the shooting method have been shown to be numerically unstable, and therefore unsuitable for applications where a level of guaranteed accuracy is required [10]. Analysis of the particular challenges associated with solving BVODEs numerically has led to the point where there are now many robust, efficient and accurate software packages for BVODEs. Due to the wide application of BVODEs and the maturity of the software available for the problem class, high-quality BVODE solvers are commonly packaged within popular high-level scientific computing environments. These include the *bvode* solver packaged in *Scilab* [11], and the *solve_bvp* solver in the Python scientific computing module *Scipy* [12].

Standard numerical methods for BVODEs make use of one-step solution procedures, where the entire solution is computed simultaneously. A simple example of a one-step method for the numerical solution of BVODEs is the Midpoint scheme. Consider the ordered partition of $[x_a, x_b]$,

$$\pi_N = \{x_a = x_1 < x_2 < ... < x_N < x_{N+1} = x_b : N \in \mathbb{N}\}. \tag{1.1}$$

The partition π_N is referred to as a mesh, and N is the number of subintervals of this mesh. Let $h_i = x_{i+1} - x_i$ be the length of the ith subinterval, $i = 1, ..., N$. When $h_i = \frac{x_b - x_a}{N}, i = 1, ..., N$, we say that π_N is a uniform mesh. The Midpoint scheme is given by

$$\frac{\underline{y}_{i+1} - \underline{y}_i}{h_i} = \underline{f}\Big(x_{i+\frac{1}{2}}, \underline{y}_{i+\frac{1}{2}}\Big), \tag{1.2}$$

where

$$\underline{y}_{i+1} \approx \underline{y}(x_{i+1}), \quad \underline{y}_i \approx \underline{y}(x_i), \quad x_{i+\frac{1}{2}} = \frac{x_{i+1} + x_i}{2},$$
$$\underline{y}_{i+\frac{1}{2}} = \frac{\underline{y}_{i+1} + \underline{y}_i}{2}, \quad i = 1, ..., N.$$

Coupling the Midpoint discretization scheme (1.2) with the boundary conditions, we obtain the following system of $n(N+1)$ non-linear equations (where n is the number of differential equations of the BVODE system)

$$\frac{\underline{y}_2 - \underline{y}_1}{h_1} - \underline{f}(x_{1+\frac{1}{2}}, \underline{y}_{1+\frac{1}{2}}) = \underline{0},$$

$$\ldots$$

$$\frac{\underline{y}_{N+1} - \underline{y}_N}{h_N} - \underline{f}(x_{N+\frac{1}{2}}, \underline{y}_{N+\frac{1}{2}}) = \underline{0},$$

$$\underline{g}(\underline{y}_1, \underline{y}_{N+1}) = \underline{0}, \tag{1.3}$$

which can be solved using standard methods for solving non-linear equations. Such solvers are employed with a sufficiently sharp tolerance to allow the error in the numerical solution of the non-linear system to be dominated by the discretization error associated with the Midpoint scheme. To improve the performance of this method and other algorithms for solving BVODEs, an initial guess for the solution is typically provided by the user to accelerate convergence when solving these systems of non-linear equations.

The Midpoint scheme is known to generate numerical solutions having $\mathcal{O}(h^2)$ accuracy [10], where $h = \max_{i=1,\ldots,N}\{h_i\}$. From this result, we see that a simple method for generating higher accuracy solutions to BVODEs using the Midpoint scheme is to reduce the subinterval sizes by adding additional mesh points. Hence a simple adaptive algorithm implementing Midpoint scheme could simply use a mesh doubling strategy, wherein the Midpoint scheme is repeatedly applied, with a uniform mesh of double the number of subintervals used on every iteration, i.e., each subinterval of the previous mesh is halved. The algorithm terminates when the difference between consecutive solutions becomes acceptable. This approach and many related algorithms are based on the use of a method called Richardson Extrapolation [13], which is used to determine an error estimate. Richardson Extrapolation for BVODEs will be described further in Chapter 2. Use of this strategy can be seen in Figure 1.1, which demonstrates the solution to the BVODE system (2.1) described in Chapter 2, as computed by a *Scilab* implementation of the Midpoint scheme on consecutive doubled meshes. In this figure, the solution becomes increasingly more accurate as N is increased. However, the size of the non-linear system associated with computing a solution with Midpoint scheme also increases as N is increased, resulting in the computation becoming more expensive. For large-scale, difficult BVODEs arising in many applications, the number of mesh subintervals required to obtain a reasonably accurate solution on a uniform mesh can be very large, and consequently, the computation can be very expensive. This motivates numerical methods which can produce more accurate solutions without changing N either at all or as little as possible. Looking at Figure 1.1, we see that even with low values for N, the solution is very accurate towards the center of the domain, allowing us to infer that fewer mesh points are required to produce an accurate solution in this area, and mesh points could instead be relocated to the difficult regions near the boundaries. Behaviour like this motivates so-called r-adaptive methods, which move mesh points to concentrate them strategically in areas where the problem is the most difficult; r-adaptive methods will be discussed in more detail in Section 1.5.

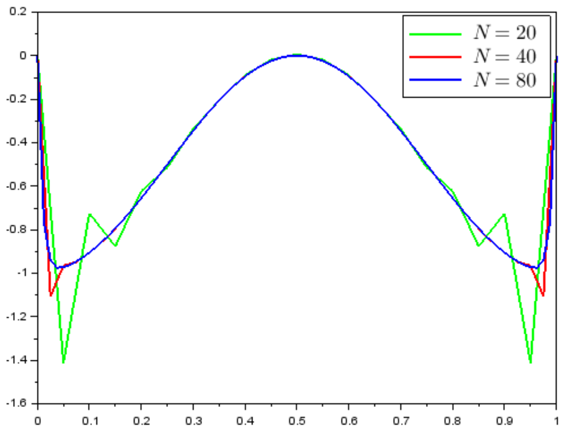


Figure 1.1: Midpoint scheme applied with different numbers of mesh subintervals.

Many discretization methods and solution procedures have been developed for BVODEs. The Midpoint scheme belongs to a class of methods known as Mono-Implicit Runge-Kutta (MIRK) methods, which include formulas attaining arbitrary orders of accuracy [10]. Continuous MIRK (CMIRK) methods are extensions of the MIRK formulas which generate continuous solution approximation in terms of a polynomial interpolant by performing a small number of additional computations [14].

Collocation methods define the approximate solution (called the collocation solution) to a BVODE in terms of piecewise polynomials. This collocation solution is determined by requiring that it satisfy the BVODE at a number of points within the domain; these are said to be the collocation conditions [10].

As mentioned earlier, the shooting method and its extensions make use of methods for IVODEs and are popular due to their relative ease of implementation and the fact that existing high-quality IVODE software can be used in their implementation [10]. Finite difference methods approximate the derivatives in the BVODE in terms of the solution at neighbouring mesh points using standard numerical differentiation formulas and are another simple, common method to solve BVODEs [10].

Standard high quality, general-purpose software for BVODEs typically implements either Runge-Kutta or collocation discretization methods. This is due to their continuous solution representation, robustness and convergence properties. An example of software using collocation methods is COLNEW [6]. An example of software implementing Runge-Kutta methods is BVP_SOLVER [15].

1.2 Overview of *bvode* / COLNEW

bvode is an adaptive error control solver for BVODEs implemented in the *Scilab* open source scientific computing environment [11]. This solver can be used to solve mixed order, multipoint BVODE systems having the form

$$\begin{aligned} y_i^{(m_i)}(x) &= f_i(x, z(y(x))), \quad i = 1, ..., n, \quad x \in [x_a, x_b], \\ g(\zeta_j, z(u(\zeta_j))) &= 0, \quad j = 1, ..., M, \end{aligned} \tag{1.4}$$

where

$$\begin{gathered} M = \sum_{i=1}^{n} m_i, \quad z(u) = \begin{bmatrix} u_1 & u_1^{(1)} & ... & u_n & ... & u_n^{(m_n-1)} \end{bmatrix}^T, \\ x_a \le \zeta_1 \le ... \le \zeta_M \le x_b. \end{gathered} \tag{1.5}$$

This solver is highly general, being able to solve many forms of BVODEs without requiring the user to perform conversions to equivalent first-order systems. This is in contrast to the Midpoint scheme (1.2), which is appropriate only for equations in first-order form.

As is the case with many solvers implemented in high-level scientific computing languages, *bvode* is simply an interface to an underlying solver written in a lower level language. For performance reasons, the equation is solved through a call to the low-level code and then the result is returned through the high-level interface, which also serves to manage appropriate memory allocation and other user convenience features. *bvode* wraps the well-known COLNEW solver [6] which is written in Fortran 77. COLNEW implements a Gaussian collocation algorithm which generates a continuous solution approximation in terms of a monomial spline basis [6]. An adaptive error control algorithm is used to compute approximate solutions having error estimates that are within a user-specified tolerance. Adaptation is accomplished by increasing the number of mesh points as well as relocating mesh points into regions having large error estimates. COLNEW has been widely applied and stands as one of the top solvers for BVODEs. Due to COLNEW's popularity and quality, it has served as a base for recent developments in the next generation of error control BVODE solvers. In [16], results are reported for a new version of COLNEW which builds off the collocation solutions generated by COLNEW in order to construct a superconvergent interpolant [17], a continuous solution approximation to the BVODE which has a global order of accuracy which is far greater than that of the collocation solution. The use of these substantially more accurate solution approximations greatly accelerates how quickly this new version of the solver can produce a solution satisfying a given error tolerance while only requiring the relatively inexpensive task of generating the superconvergent interpolant.

bvode will be used in our experimentation to provide high accuracy solutions to some of the problems considered. Additionally, the ability to request different levels of error tolerance provides the opportunity to experiment with the role numerical error at different levels can play in the computations.

1.3 Numerical Methods for PDEs

Due to their fundamental role in mathematical modelling, numerical methods for PDEs have been of great interest to both application domain experts and in numerical analysis. This has motivated a large volume of research into numerical methods and software for this problem class. With the presence of multiple independent variables, the problem of solving PDEs involves, compared to BVODEs, greater theoretical difficulty, more complicated numerical methods, and greater computational expense than when solving ODEs in order to obtain numerical solutions that have a reasonable level of accuracy. This makes the development of reliable software for general classes of problems difficult, with the majority of software being developed for specific application problems. This is particularly common in the case of time-dependent PDEs with two or more spatial dimensions, where, to our knowledge, no high-quality error control solvers have been developed.

While some packages for solving general classes of 1D PDEs are available within environments such as MATLAB and Maple, they typically do not provide full control of the solution error. Typically, these packages only control the error contributed to the solution approximation from the temporal domain, with no control of the error contributed from the spatial domain. For the 1D time-dependent PDE case, the BACOL package was among the first general-purpose PDE solver which provided both spatial and temporal error control [18]. The BACOLR package was later developed; it implements a modification of the BACOL algorithm which improves its performance for certain classes of PDEs [19]. The recently developed BACOLI package [7] substantially improves the efficiency of the BACOL algorithm through the use of efficient interpolation-based spatial error estimation. Recent developments in this area include the BACOLRI package [16], which is a modification of BACOLR to include the algorithmic improvements of BACOLI and has been shown to have superior performance to BACOLI for certain classes of problems.

A standard numerical method for solving time-dependent 1D PDEs is the Method of Lines [20]. In this approach, a numerical discretization method is applied to the spatial variables which leads to the approximation of the PDE by a system of time-dependent ODEs [20]. The numerical discretization method applied in the spatial domain is often similar to those used to solve BVODEs. The time-dependent ODE system that approximates the PDE is then solved using a standard IVODE solver to obtain the solution at the next point in time. In this way, the Method of Lines can be thought of as combining numerical methods for BVODEs and IVODEs by alternating the treatment of the spatial and the temporal components of the solution. We next describe a simple Method of Lines algorithm which can be easily implemented in scientific programming languages such as *Fortran* or *Scilab.*

Consider the uniform mesh of N subintervals,

$$\pi_N = \{x_a = x_1 < x_2 < ... < x_N < x_{N+1} = x_b : N \in \mathbb{N}\}. \tag{1.6}$$

Then the following finite difference discretization methods provide $\mathcal{O}(h^2)$ approximations for the first and second spatial partial derivatives at the each of the internal mesh points $x_i, i = 2, ..., N$, where h is the uniform subinterval length in π_N [21],

$$\underline{u}_x(x_i, t) \approx \frac{\underline{u}_{i+1}(t) - \underline{u}_{i-1}(t)}{2h}, \tag{1.7}$$

$$\underline{u}_{xx}(x_i, t) \approx \frac{\underline{u}_{i+1}(t) - 2\underline{u}_i(t) + \underline{u}_{i-1}(t)}{h^2}. \tag{1.8}$$

Here $\underline{u}_i(t) \approx \underline{u}(x_i, t)$, where $u(x, t)$ is the exact solution to (3). We approximate the first derivatives at the spatial boundaries x_a, x_b the using one-sided, second order, finite difference schemes [21]

$$\underline{u}_x(x_a, t) \approx \frac{-3\underline{u}_1(t) + 4\underline{u}_2(t) - \underline{u}_3(t)}{2h}, \quad \underline{u}_x(x_b, t) \approx \frac{\underline{u}_{N-1}(t) - 4\underline{u}_N(t) + 3\underline{u}_{N+1}(t)}{2h}. \tag{1.9}$$

Substituting the approximations (1.7) and (1.8) into the general form of the PDE, we obtain

$$\underline{u}_t(x_i,t) \approx \underline{f}\left(x_i, t, \underline{u}_i(t), \frac{\underline{u}_{i+1}(t) - \underline{u}_{i-1}(t)}{2h}, \frac{\underline{u}_{i+1}(t) - 2\underline{u}_i(t) + \underline{u}_{i-1}(t)}{h^2}\right). \tag{1.10}$$

This gives us a system of IVODEs, one at each mesh point, with the initial values given by the known value of the solution at the current time t. At the time t_0, the initial conditions (5) are used to provide the required solution information for the IVODEs. Combining this system with the associated boundary conditions (4) and using (1.9), we obtain

$$\underline{b}_L\left(t, \underline{u}_1(t), \frac{-3\underline{u}_1(t) + 4\underline{u}_2(t) - \underline{u}_3(t)}{2h}\right), \quad \underline{b}_R\left(t, \underline{u}_{N+1}(t), \frac{\underline{u}_{N+1}(t) - 4\underline{u}_N(t) + 3\underline{u}_{N-1}(t)}{2h}\right). \tag{1.11}$$

Together (1.10) and (1.11) give a system of Differential Algebraic Equations (DAEs) which can be solved using typical methods for DAEs. It is worth noting that the accuracy of the approximate solution produced by this method is bounded by the $\mathcal{O}(h^2)$ error contribution of the spatial discretization. To make the overall error depend solely on this discretization error, we require that a DAE solver solve the DAE system to a level of accuracy greater than that which is associated with the spatial discretization. The advantage of doing so is that the contributions to the error by the spatial discretization and time integration can be considered separately, making it easier to control both sources of error. To achieve a sufficiently accurate solution to the DAEs, standard error control software such as DASSL [22] or RADAU5 [23] can be applied with an error tolerance which is slightly lower than the error desired for the overall computation. In high-level computing environments, examples of error control DAE solvers suitable for this purpose include *Scilab's dae* solver [24], or the *ode15i* solver in MATLAB [25].

An algorithm, written in pseudocode, for the Finite Difference Method of Lines approach described above is given in Algorithm 1. We have implemented this algorithm in MATLAB using the finite difference spatial discretizations (1.7)-(1.9), with *ode15i* used to compute an error-controlled solution to the resultant DAE systems. This implementation of Algorithm 1 was used to solve the

PDE problem (3.1) with initial conditions (3.2), boundary conditions (3.3) and (3.4), with problem parameter $\epsilon = 10^{-2}$, using varying choices of N for a uniform spatial mesh and DASSL with tolerances of 10^{-8}. The results of this computation are given in Figure 1.2. From this figure, we see the effect of the spatial error on the numerical solution to the PDE, with high spatial errors at any time step being able to propagate forward in time and possibly contaminate the rest of the computation. Since the temporal error in a Method of Lines algorithm can be controlled simply by using standard high-quality DAE solvers, which can efficiently adapt the computation of the temporal component of the numerical solution, control of the spatial error is the factor which is of primary interest when developing error control algorithms for PDEs. This indicates important similarities between the adaptive methods for BVODEs and PDEs, with each case requiring methods which can reduce the spatial errors occurring in the discretization of the problem.

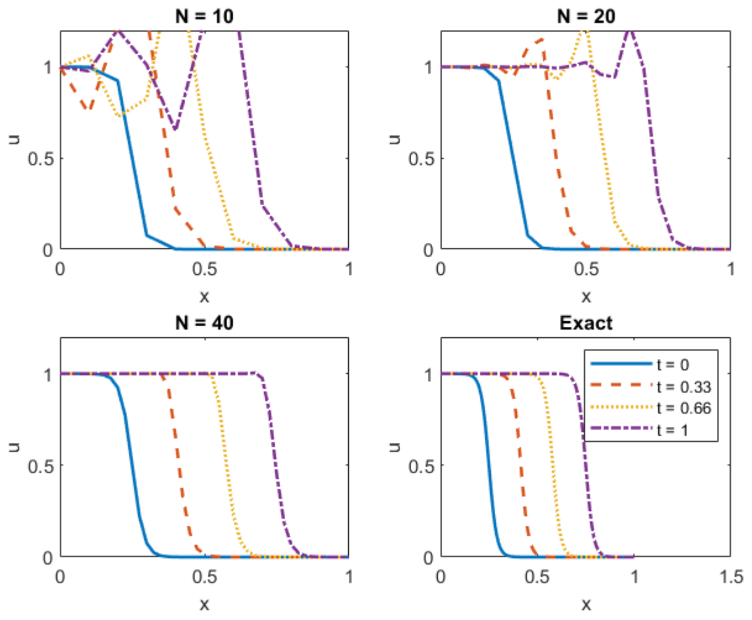


Figure 1.2: Finite Difference Method of Lines algorithm, applied with increasing N values. Plotted against the exact solution.

Finite difference methods such as the second-order formula given in this section are often used for discretizing PDEs due to the ease of programming such methods. The Finite Element method is the standard method for areas such as Engineering and Mathematical Physics for solving problems having complex geometries, particularly for problems having more than one spatial dimension [20].

Algorithm 1: Method of Lines with Finite Difference Discretization

```
function MOL ($x$, $t_0$, $t_{out}$, $PDE$, $uinit$, $bndxa$, $bndxb$);
Input  : Mesh $x$; start time $t_0$; problem definition $PDE$; initial condition $uinit$; left BC $bndxa$;
         right BC $bndxb$
Output: Solution to the PDE at time $t_{out}$, $u_{out}$
$u_1 := bndxa(t_0)$
$u_{2,\ldots,N} := uinit(x_{2,\ldots,N})$
$u_{N+1} := bndxb(t_0)$
$t := t_0$
while $t < t_{out}$ do
    $daeSys := FiniteDiff(x, u, PDE, bndxa, bndxb)$
    $u(x, t+s) := Integrate(t, daeSys)$
    $t := t + s$
end
$u_{out} := u(x, t_{out})$
return $u_{out}$
```

As in the case of BVODEs, collocation methods have been a common choice when developing high-quality packages for 1D PDEs as they can be used to generate high order, continuous solution approximations [18, 19, 7].

1.4 Overview of BACOLI

Since one of the primary goals of this thesis is to explore adaptive methods for PDEs which can be applied in developing PDE solvers with error control, it is worth briefly describing the current state of the art in this area. The BACOL family of software packages are a collection of 1D PDE solvers featuring high-order numerical methods combined with an adaptive error control algorithm which have been efficiently implemented in Fortran 77, with Fortran 95 wrappers for the newest members of this family [16]. Each solver in this family uses an Adaptive Method of Lines (AMOL) algorithm in order to solve Initial-Boundary Value problems of the form (3), with the most recent and efficient package being BACOLI [7, 16]. AMOL algorithms are Method of Lines algorithms which adapt the spatial discretization component of the method in addition to the standard temporal adaptation. This allows for control over both the spatial and temporal errors.

BACOLI makes use of a continuous solution approximation, represented in terms of a linear combination of C^1-continuous B-spline basis functions $\{B_{i,p}(x)\}_{i=1}^{NC_p}, NC_p = N(p-1)+2$, each of user-specified degree p. These B-spline basis functions are implemented using the de Boor B-spline package [26]. At any time $t > t_0$, the approximate solution can be written as

$$\underline{U}(x,t) = \sum_{i=1}^{NC_p} \underline{y}_{p,i}(t) B_{p,i}(x), \tag{1.12}$$

where $\underline{y}_{p,i}(t)$ is an unknown time-dependent coefficient for the ith B-spline basis function. From this expression we can obtain the expressions for the first two spatial partial derivatives,

$$\underline{U}_x(x,t) = \sum_{i=1}^{NC_p} \underline{y}_{p,i}(t) B'_{p,i}(x), \tag{1.13}$$

$$\underline{U}_{xx}(x,t) = \sum_{i=1}^{NC_p} \underline{y}_{p,i}(t) B''_{p,i}(x). \tag{1.14}$$

To determine the coefficients $y_{p,i}(t)$ we first require that the approximate solution satisfy the PDE at certain points within each subinterval. These conditions are known as the collocation conditions and the points at which they are imposed are called the collocation points. Let $\{h_i\}_{i=1}^{N}$ be the size of the ith subinterval, i.e., let $h_i = x_i - x_{i-1}$, and let $\{\rho_i\}_{i=1}^{p-1}$ be the set of order $p-1$ Gauss points on $[0,1]$ such that

$$0 < \rho_1 < ... < \rho_{p-1} < 1. \tag{1.15}$$

Then the collocation points are chosen to be

$$\eta_1 = a, \quad \eta_l = x_{i-1} + h_i \rho_j, \quad \eta_{NC_p} = b, \tag{1.16}$$

with $l = 1 + (i-1)(p-1) + j, i = 1, ..., N$, and $j = 1, ..., p-1$.

At each point in the sequence of $NC_p - 2$ collocation points, $\{\eta_i\}_{i=2}^{NC_p - 1}$, we substitute evaluations of the approximate solution representation (1.12) and its derivatives (1.13) and (1.14) into the PDE

(3), giving the collocation conditions,

$$\frac{d}{dt}\underline{U}(\eta_l,t) = \underline{f}(t,\eta_l,\underline{U}(\eta_l,t),\underline{U}_x(\eta_l,t),\underline{U}_{xx}(\eta_l,t)),$$
$$l = 2,...,NC_p - 1. \tag{1.17}$$

This system combined with the BCs at points $\eta_1 = a$ and $\eta_{NC_p} = b$,

$$\underline{b}_L(t,\underline{U}(a,t),\underline{U}_x(a,t)) = \underline{0}, \quad \underline{b}_R(t,\underline{U}(b,t),\underline{U}_x(b,t)) = \underline{0}, \tag{1.18}$$

gives a system of DAEs which is then solved using a slightly modified version of DASSL with an error tolerance which is slightly more stringent than the user-prescribed error tolerance to ensure that the error contributed from spatial discretization dominates the error of the temporal computation. After DASSL solves the system one step forward in time, a spatial error estimate is computed for the solution at this new time step. This spatial error estimate is obtained by using the current values of the solution at certain points within each subinterval to construct either a Superconvergent Interpolant - which is a solution approximation of one greater order of accuracy than the computed approximate solution (1.12), and which is used for performing standard error estimation - or a Lower Order interpolant - which is a solution with order of accuracy one less than (1.12), which is used for computing local extrapolation-based error estimates [7]. Further details are given in [7, 27]. We then have the current approximation $\underline{U}(x,t_{i+1})$ and another approximate solution $\underline{\bar{U}}(x,t_{i+1})$ which differs by an order of accuracy. We then compute the global scaled error estimates over the whole spatial domain, one for each PDE,

$$E_s(t) = \sqrt{\int_{x_a}^{x_b}\left(\frac{U_s(x,t) - \bar{U}_s(x,t)}{ATOL_s + RTOL_s|U_s(x,t)|}\right)^2 dx}, \quad s = 1,...,NPDE, \tag{1.19}$$

and N error estimates, one for each subinterval,

$$\hat{E}_i(t) = \sqrt{\int_{x_i}^{x_{i+1}} \left(\frac{U_s(x,t) - \bar{U}_s(x,t)}{ATOL_s + RTOL_s|U_s(x,t)|} \right)^2 dx}, \quad i = 1, ..., N, \tag{1.20}$$

where $ATOL$ and $RTOL$ are user-specified values for the absolute and relative error tolerances. If the scaled global error estimates (1.19) are less than 1, then the computed solution at the current time step is accepted and the algorithm is repeated at the next time step. Otherwise, the step is rejected and a new spatial mesh of N^* subintervals is generated, with the locations of the mesh points chosen so that more points are clustered where the spatial error estimates (1.20) are large, and the algorithm is repeated from the previous time step. N^* is chosen to attempt to obtain a new numerical solution whose estimated error is less than the user tolerance. See [28] for details.

BACOLI has been shown experimentally to efficiently produce solutions which are accurate to within a user tolerance. The code has been applied to a variety of model problems from several application domains [29, 30]. BACOLI's algorithm serves as a basis for later discussion of error control algorithms for PDEs.

1.5 Adaptation Strategies and the Moving Transformation Method

Similarities between the numerical methods used for BVODEs and 1D PDEs allow us to apply similar methods of solution adaptation to both problem classes. In the PDE context, the Method of Lines allows us to focus on adaptation in the spatial domain, with temporal adaptation done using existing IVODE solvers. Therefore, the controllable factors common to both BVODEs and PDEs that are useful in adaptive algorithms depend on the error introduced within the spatial discretization process, or alternatively, indicators of locations of relative difficulty in the spatial domain. These adaptation strategies are partitioned into three main classes.

The p-refinement strategy adapts the computation by increasing the order of the discretization method, either globally across the whole spatial domain or locally within each subinterval [5]. For example, high-quality solvers implementing collocation methods for BVODEs and PDEs typically allow a range of values for p, the order of the spline basis functions, with higher p values corresponding to solution approximations with higher orders of accuracy [7, 6]. Adaptive p-refinement collocation algorithms could adaptively switch between p values, with p chosen to be small when the error due to spatial discretization is low, and chosen to be large when the error estimate is large, though to our knowledge, no solvers implementing such a method exist.

A popular adaptation strategy is the h-refinement approach, which improves the accuracy of spatial discretization by adding additional mesh points in regions of high solution error [5]. This is typically implemented by adding an additional mesh point at the midpoint of subintervals where the error estimate is large.

An alternative adaptive strategy, r-refinement, which typically works by relocating the existing mesh points to areas where the solution error is large [4], is the main focus of this thesis. In high-quality solvers, a hybrid of these strategies is typically implemented since the use of only one method can be potentially ineffective or inefficient depending on the given problem. The hr-refinement methods implemented in software such as COLNEW and BACOLI vary the number of subintervals and reposition the mesh points in order to produce sufficiently accurate numerical solutions which meet a user-specified error tolerance [6, 7], using the minimum number of mesh points.

The majority of r-adaptive methods control the location of mesh points through a computational device called the equidistribution principle [4]. In the 1D case, the equidistribution principle prescribes a unique mesh which equidistributes a monitor function $M(x) > 0$, generally chosen to be an indicator of quantities such as solution error or areas of rapid variation in the solution [4]. The equidistribution principle prescribes that the mesh satisfy

$$\int_{x_a}^{x_{i+1}} M(x)dx = \frac{i}{N}\sigma, \quad i = 1, ..., N, \tag{1.21}$$

where

$$\sigma = \int_{x_a}^{x_b} M(x)dx. \tag{1.22}$$

A mesh which satisfies the above condition is said to be an *equidistributing mesh.* Figure 1.3 gives the example of the function $f(x) = 5x+1$ partitioned on both a uniform mesh and equidistributed mesh, demonstrating that when using the equidistributed mesh, the area associated with each subinterval is the same.

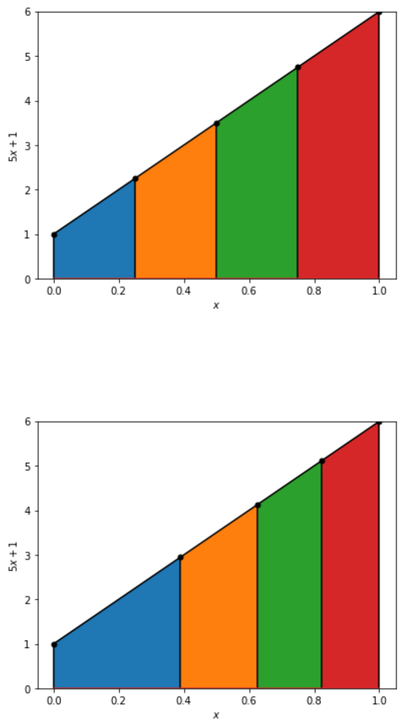


Figure 1.3: $f(x) = 5x + 1$ on uniform (top) and equidistributed (bottom) meshes.

In general, computing an equidistributing mesh must be done using a numerical method [4]. A standard numerical method used to generate an approximately equidistributed mesh is de Boor's algorithm [31]. This algorithm simplifies the task of computing the integrals in (1.21) numerically by approximating the monitor function as a piecewise constant function and equidistributing this simplified form. Algorithm 2 gives de Boor's algorithm. For practical applications de Boor's algo-

Algorithm 2: de Boor's Algorithm [4]

```
function deBoor $(x_a,\ x_b,\ N,\ \bar{x},\ m)$;
Input  : Spatial boundaries $x_a, x_b$; number of subintervals $N$; initial background mesh $\bar{x} = \{\bar{x}_i\}_{i=1}^{N+1}$;
         $m = \{M(\bar{x}_i)\}_{i=1}^{N+1}$
Output: Equidistributed mesh $x = \{x_i\}_{i=1}^{N+1}$
$M := N + 1$
$\sigma := 0$
for $i = 2$ to $M$ do
    $\sigma := \sigma + (\bar{x}_i - \bar{x}_{i-1})(m_i + m_{i-1})$
end
$\sigma := \frac{\sigma}{2}$; $x_1 := \bar{x}_1$; $x_M := \bar{x}_M$; $k := 2$; $a := 0$
$\epsilon_{MACH}$ := machine epsilon
for $i = 2$ to $M$-1 do
    $\sigma_i := \bar{x}_i \sigma$
    for $j = k$ to $M$ do
        $b := a + 0.5(\bar{x}_j - \bar{x}_{j-1})(m_j + m_{j+1})$
        if $\sigma_i < b + \epsilon_{MACH}$ then
            $k := j$
            break
        end
        $a := b$
    end
    $x_i := \bar{x}_{k-1} + \frac{2(\sigma_i - a)}{m_k + m_{k-1}}$
end
return $x$
```

rithm has been shown to be efficient and robust, generating meshes which are sufficiently close to satisfying the equidistribution principle [4].

While the equidistribution principle is ubiquitous in 1D r-refinement algorithms due to its optimality properties and efficiency, it is not sufficient to specify a unique grid for problems with multiple spatial dimensions [4, 5]. Additional constraints must be imposed in the higher dimensional cases. Examples of such constraints include conditions on the alignment of grid elements and optimal transport conditions [4, 5].

To develop an alternative to r-adaptation based on equidistribution-based mesh generation, it is useful to generalize the equidistribution principle to an equivalent continuous form. This is done by considering the locations of the mesh points to be given under the image of a coordinate transformation, $x(\xi)$, that maps from a computational domain $\Omega_c = [0, 1]$ to the physical domain $\Omega_p = [x_a, x_b]$ on which the problem is defined, i.e.,

$$x : \Omega_c = [0, 1] \mapsto \Omega_p = [x_a, x_b], \tag{1.23}$$

such that the ith mesh point in Ω_p, x_i, is given by

$$x_i = x(\xi_i), \quad i = 1, ..., N+1, \tag{1.24}$$

where

$$\xi_i = \frac{i-1}{N}, \quad i = 1, ..., N+1, \tag{1.25}$$

are the mesh points of a uniform spatial mesh on Ω_c [4]. With this coordinate transformation, the equidistribution principle (1.21) can be rewritten as [4]

$$\int_{x_a}^{x(\xi_i)} M(x(\xi_i))dx = \sigma\xi_i, \quad i = 1, ..., N+1. \tag{1.26}$$

To generalize the equidistribution principle as a fully continuous problem, the coordinate transformation, $x(\xi)$, is defined to be an *equidistributing coordinate transformation* if it satisfies the condition [4]

$$\int_{x_a}^{x(\xi)} M(x(\xi))dx = \sigma\xi, \quad \forall\xi \in [0,1]. \tag{1.27}$$

In this continuous generalization, we see that the problem of adapting the spatial mesh becomes an issue of generating an equidistributing coordinate transformation $x(\xi)$ as defined in (1.27), with the discrete mesh generation case (1.26) being recovered when we simply evaluate $x(\xi)$ at the uniform mesh points on Ω_c, (1.25).

Twice differentiating (1.27) with respect to ξ, we have

$$\frac{d}{d\xi}\left(M(x(\xi))\frac{dx(\xi)}{d\xi}\right) = 0. \tag{1.28}$$

When coupled with the BCs

$$x(0) = x_a, \quad x(1) = x_b, \tag{1.29}$$

the differential equation (1.28) gives us the ability to solve for the coordinate transformation $x(\xi)$ using typical methods for BVODEs. We will refer to this equation as MMODE0.

From this continuous generalization of standard approaches for r-adaptation, we can derive the Moving Transformation (MT) adaptive methods. The MT approach provides a framework through which r-adaptivity can be used to generate accurate, adaptive solutions to BVODEs and PDEs. In particular, *it allows us to consider the transformation of a differential equation itself onto the computational space Ω_c such that the solution of the transformed differential equation is smoothed in areas where the monitor function $M(x(\xi))$ is large* [4]. Since $M(x(\xi))$ typically represents quantities such as solution error or rapid change in the solution, transforming the differential equation to this smoothed domain allows the accurate computation of a numerical solution to this transformed differential equation on a uniform computational mesh in Ω_c while using relatively few mesh points [4]. To simplify our notation, for the remainder of this chapter, we assume without loss in generality that $\Omega_c = \Omega_p$. That is, we assume $x_a = 0$ and $x_b = 1$.

The solution to (1.28) must be solved approximately when using a monitor function $M(x(\xi))$ which is not known *a priori*, but for practical purposes, the transformation need not be computed to a particularly high level of accuracy [4] and therefore MMODE0 can be solved reasonably efficiently. However, there are certain properties that any numerical method used to solve (1.28) must preserve. The first condition which must hold is the monotonicity of the coordinate transformation; that is, if $\xi_i > \xi_j$, $\xi_i, \xi_j \in \Omega_c$, then we must have $x(\xi_i) > x(\xi_j)$ [4]. Another important property that must hold is that the values at the boundaries must be preserved, i.e., we must have that $x(0) = 0$ and $x(1) = 1$ [4]. The difficulty in solving an MT equation such as (1.28) is related to how rapidly $M(x(\xi))$ varies over the spatial domain [4].

To transform the differential equation into the computational space, we must rewrite it in terms of the independent variable in the computational space, ξ [4]. This involves coupling the differential equation with the derivatives of the coordinate transformation. In the case of BVODEs, we couple the equation with the invertible coordinate transformation $x(\xi)$ by first defining the transformed solution to the BVODE, $\underline{\hat{y}}(\xi)$, as follows

$$\underline{\hat{y}}(\xi) := \underline{y}(x(\xi)). \tag{1.30}$$

Differentiating with respect to ξ,

$$\begin{aligned} \frac{d}{d\xi}\underline{\hat{y}}(\xi) &= \frac{d}{dx}\underline{y}(x(\xi))\frac{d}{d\xi}x(\xi), \\ &= \underline{f}(x(\xi), \underline{y}(x(\xi)))\frac{d}{d\xi}x(\xi), \\ &= \underline{f}(x(\xi), \underline{\hat{y}}(\xi))\frac{d}{d\xi}x(\xi). \end{aligned} \tag{1.31}$$

This gives us the transformed differential equation on the computational domain. Combining (1.31) and (1.28), we have the coupled, non-linear BVODE system, on Ω_c,

$$\frac{d}{d\xi}\underline{\hat{y}}(\xi) = \underline{f}(x(\xi), \underline{\hat{y}}(\xi))\frac{d}{d\xi}x(\xi), \tag{1.32}$$

$$\frac{d}{d\xi}\Big(M(x(\xi))\frac{dx(\xi)}{d\xi}\Big) = 0, \tag{1.33}$$

with the associated boundary conditions

$$\underline{\hat{g}}(\underline{\hat{y}}(0), \underline{\hat{y}}(1)) = \underline{0}, \tag{1.34}$$

$$x(0) = 0, \quad x(1) = 1. \tag{1.35}$$

We note that the differential equation (1.32) is dependent on the solution to (1.33), but (1.33) has no dependence on the solution to (1.32). Therefore the coupling in the system (1.32-1.33) can be eliminated by first solving (1.33), and then using the resultant coordinate transformation to transform and then solve (1.32) on Ω_c. While the simultaneous solution procedure, where the coupling of (1.32) and (1.33) is treated directly, is perhaps the more natural approach, the alternating procedure, which separates these two components of the computation, allows us to remove the additional difficulties that the coupling introduces, meaning that both the coordinate transformation and the transformed BVODE can be solved efficiently [4]. Solution procedures are further discussed in Chapter 2.

In the PDE case, we require that the coordinate transformation depends continuously on time, i.e., we use a coordinate transformation $x(\xi, t)$ rather than $x(\xi)$ [4]. This allows the coordinate transformation to adapt to the behaviour of a time-dependant monitor function $M(x(\xi, t), t)$ that describes the regions of solution difficulty to a PDE, which may of course change as the PDE is solved forward in time. The time-dependent generalization of MMODE0, MMPDE0, is given by

$$\Big(M(x(\xi, t), t)x_\xi(\xi, t)\Big)_\xi = 0, \tag{1.36}$$

which perscribes that at each point in time, the coordinate transformation must satisfy the continuous form of the equidistribution principal (1.27) [4]. However, in the PDE case, it is common to introduce explicit dependence on the time derivatives of the coordinate transformation [4, 5], with an example being the popular MMPDE5, given in [4] as,

$$x_t(\xi,t) = \frac{1}{\tau}\Big(M(x(\xi,t),t)x_\xi(\xi,t)\Big)_\xi, \quad \tau > 0. \tag{1.37}$$

MMPDE5 converges to MMPDE0 in its steady state, with the speed at which its solution converges to equidistribution being determined by the choice for the relaxation parameter τ, which is typically chosen to be a small, positive number [4]. The advantage of using MMPDEs with explicit dependence on the time derivatives is that the solution to this PDE, i.e., the coordinate transformation, converges smoothly over time to the equidistributing coordinate transformation. This can improve the accuracy of the solution to the transformed physical PDE in cases where the monitor function changes very rapidly [4]. If τ is chosen to be too large, then the transformation will converge very slowly to an equidistributed state and therefore the mesh will be poorly adapted to the solution of the transformed physical PDE; if τ is chosen to be too small then the DAEs resulting from the spatial discretization become stiff and are therefore more difficult to solve accurately; however, we can be reasonably sure that the physical PDE will be transformed sufficiently well so that it can be solved easily in Ω_c [4, 5]. We discuss the choice of this parameter in greater detail in Chapter 3. Many alternative MMPDEs have been derived; see [4] for further discussion.

To transform a general PDE to the computational domain, we first define the transformed solution to the PDE, $\underline{\hat{u}}(\xi,t)$, as follows,

$$\underline{\hat{u}}(\xi,t) := \underline{u}(x(\xi,t),t). \tag{1.38}$$

Differentiating with respect to ξ and $\xi\xi$ and t, we have

$$\underline{u}_t(x(\xi,t),t) = \underline{\hat{u}}_t(\xi,t) - \frac{\underline{\hat{u}}_\xi(\xi,t)}{x_\xi(\xi,t)} x_t(\xi,t),$$

$$\underline{u}_x(x(\xi,t),t) = \frac{1}{x_\xi(\xi,t)} \underline{\hat{u}}_\xi(\xi,t) \tag{1.39}$$

$$\underline{u}_{xx}(x(\xi,t),t) = \frac{1}{x_\xi(\xi,t)} \left(\frac{\underline{\hat{u}}_\xi(\xi,t)}{x_\xi(\xi,t))} \right)_\xi . \tag{1.40}$$

This allows us to write the transformed PDE in the computational domain as

$$\underline{\hat{u}}_t(\xi,t) = \underline{f}\left(x(\xi,t), t, \underline{\hat{u}}(\xi,t), \frac{\underline{\hat{u}}_\xi(\xi,t)}{x_\xi(\xi,t)}, \frac{1}{x_\xi(\xi,t)} \left(\frac{\underline{\hat{u}}_\xi(\xi,t)}{x_\xi(\xi,t)} \right)_\xi \right) + \frac{\underline{\hat{u}}_\xi(\xi,t)}{x_\xi(\xi,t)} x_t(\xi,t), \tag{1.41}$$

with the transformed boundary conditions

$$\underline{b}_L\left(t, \underline{\hat{u}}(\xi_a,t), \frac{1}{x_\xi(\xi,t)} \underline{\hat{u}}_\xi(\xi_a,t) \right) = \underline{0}, \tag{1.42}$$

$$\underline{b}_R\left(t, \underline{\hat{u}}(\xi_b,t), \frac{1}{x_\xi(\xi,t)} \underline{\hat{u}}_\xi(\xi_b,t) \right) = \underline{0}.$$

When the coordinate transformation is sufficiently adapted to the solution of the physical PDE, the transformed physical PDE can be effectively solved on Ω_c using a uniform mesh [4]. This fact is particularly beneficial for higher-dimensional problems as it avoids having to use complicated data structures to preserve the connections between neighbouring non-uniform grid points. As in the case of MT methods for BVODEs, the coupled system of the MMPDE and the transformed PDE can be solved using either a simultaneous or alternating procedure, with the alternating procedure having similar positive impacts on the performance of the overall computation [4, 5]. In the BVODE case, the alternating and simultaneous solution procedures are largely equivalent; in the PDE case, there is more to consider, as the additional time dependence in the coordinate transformation can cause the coordinate transformation to lag behind the solution, potentially leading to a degradation in the quality of the adaptivity [4, 5]. If the adaptivity is poor in an r-adaptive method then more mesh points will be required in order to accurately discretize the differential equation, which can have a substantial cost in terms of the efficiency of the algorithm.

MT methods for time-dependent PDEs are broadly categorized into two classes, quasi-Lagrange approaches, and rezoning approaches [4]. These two approaches are differentiated by how the movement of the coordinate transformation and the solution to the physical PDE through time are coordinated. In the quasi-Lagrange approach, the coordinate transformation is considered to move continuously with the physical PDE as its solution is advanced through time [4]. The rezoning approach considers the coordinate transformation to move independently of the solution to the physical PDE [4]. These two approaches are depicted graphically in Figure 1.4. The quasi-Lagrange approach can be applied using either simultaneous or alternating solution procedures, whereas the rezoning approach necessitates the use of an alternating procedure. Further details on these solution procedures are given in Chapter 3.

The relationships between the solution information contained on the computational and physical spaces are demonstrated in Figure 1.5, where U is the physical solution on Ω_p, M is the monitor function on Ω_p, $\hat{U}$ its smoothed computational analog on Ω_c, and $\hat{M}$ the smoothed monitor function on the computational domain, which is useful in practice for certain MT algorithms. Note that $\xi(x,t) := x^{-1}(\xi, t)$.

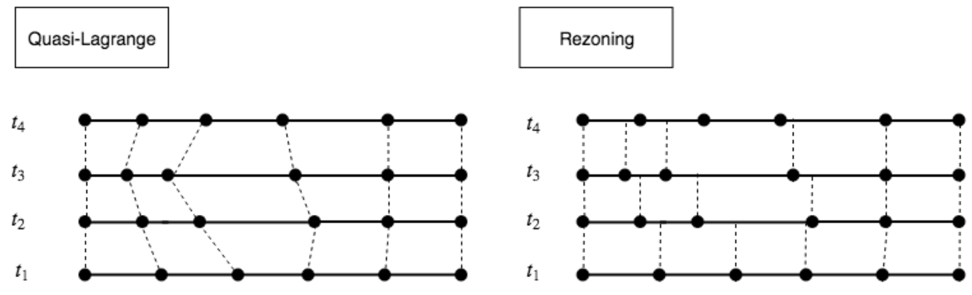


Figure 1.4: Quasi-Lagrange and rezoning MT approaches.

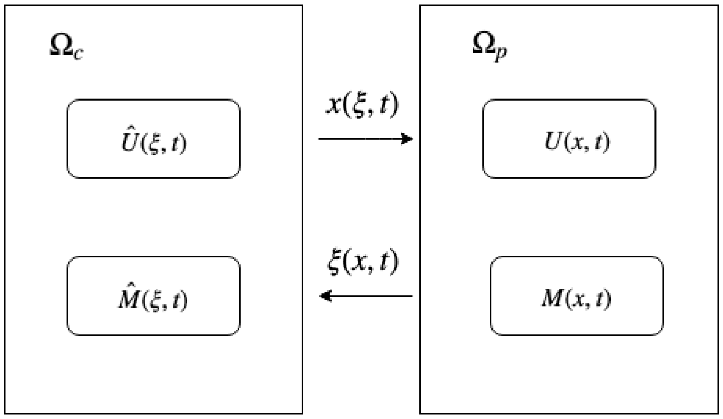


Figure 1.5: Relationship between Ω_c and Ω_p.

The majority of the research into and application of MT methods have been for PDE problems, as opposed to BVODEs. This is due to the fact that r refinement in the BVODE can be implemented far more efficiently by simply discretizing the physical differential equation on a directly computed, approximately equidistributing mesh using Algorithm 2. While this is also true in the 1D PDE case, these 1D MT methods have more natural generalizations to the multivariate case due to the time dependence of the coordinate transformation, and therefore have been of substantial research interest. Regarding the research that has been done on these methods, the emphasis has been on the mathematical analysis of coordinate transformation generation techniques, choice of monitor functions and choice of discretization method, which has produced a foundation upon which robust solvers implementing MT methods or related algorithms such as MOVCOL have been developed [32]. We do not consider MOVCOL to be an implementation of an MT method because it does not transform the PDE being solved onto the computational domain using a coordinate transformation; rather it uses the coordinate transformation obtained as the solution to an MMPDE to generate an equidistributed mesh on which the PDE is discretized on Ω_p. The majority of software implementing these methods use low order numerical methods which produce discrete solution values at only the mesh points, do not adapt the number of mesh points used, and are frequently application specific. Additionally, to our knowledge, no software which offers error control with this algorithm class has been developed.

It is therefore the goal of this thesis to explore MT approaches which can be adapted for use in general-purpose, production level numerical solvers that feature adaptive error control. The approaches considered here are of course not the most efficient approaches to use in the 1D problem classes considered, where the mesh generation problem is well defined and well-adapted meshes can be obtained efficiently. This exploration is therefore motivated by the possible application of generalizations of these approaches to higher-dimensional problems, where robust r-adaptation algorithms are substantially more difficult to derive. Tensor product B-splines Gaussian collocation software such as BACOL2D [9] will be a natural area of application of MT methods, as this code produces high-order continuous solution approximations on uniform 2D grids. When coupled with an MT method, this code could potentially produce efficient, high accuracy solution approximations to 2D PDEs, based on an extension of this code to use an efficient adaptive error control algorithm.

Chapter 2

Moving Transformation Methods for BVODEs

MT methods have received a great deal of attention and study for the purpose of adaptively solving difficult multi-dimensional PDEs [4, 5]; however, the application of these methods in adaptive error control algorithms has not been well explored. Existing MT implementations have typically been applied to time-dependent PDEs, with the motivation for their use primarily being to adapt the computation such that the behaviour of the approximate solution is sufficiently close to the behaviour of the physical system being modelled. This is in contrast to the approach taken in adaptive error control algorithms, where the goal is always to generate a solution whose estimated error is within a user-prescribed tolerance. Of course, when the error is sufficiently small the solution computed using an error-control solver will necessarily match the physical characteristics of the differential equation, giving the error control approach significant robustness. In order to implement error control, three factors must be considered: a scheme for estimating the error, an adaptation procedure for refining the solution approximation in regions of high estimated error, and an iterative refinement procedure for repeatedly improving the solution until the estimated error is within the user-provided tolerance.

In this chapter, we explore MT methods applied to BVODEs in order to understand the role that these methods can play as the adaptation procedure in an adaptive error control algorithm. Our interest in applying MT methods to BVODEs is to provide a minimal example of the use of these methods, allowing us to more easily analyze the capabilities of these methods and the computational challenges which arise in their use. MT methods for BVODEs have received little attention in the literature, likely due to existing direct, robust r-adaptive algorithms available for these problems such as de Boor's algorithm (Algorithm 2), which MT algorithms are unlikely to match in terms of efficiency. Some mention is given in [10] but is largely presented as a tool for use in theoretical analysis rather than practical application. In [4], discretizations of MMODE0 are presented in the context of mesh generation; however, there is no discussion of its use in an MT method. In our exploration, we first consider the application of MT adaptation without iterative refinement and error control to demonstrate the effects of MT adaptation. This leads to further discussion and experimentation with iterative refinement procedures for BVODEs with error estimation and control using MT methods.

Since we wish to consider methods that will help to inform the development of error control algorithms, our goal to eventually use MT adaptivity which is driven using a monitor function based on an error estimate. This will allow the computation to be adapted to regions of the domain where the estimated error in the solution is large. When implementing error control for the problem classes considered in this thesis, many modern, standard approaches make use of error estimates which can be expressed in terms of a polynomial interpolant [16]. Hence an essential component of this work is to implement MT methods where the adaptivity is driven through the use of an interpolated monitor function. We explain in Section 2.1 how the monitor function is coupled with an interpolant.

This chapter is organized as follows: Section 2.1 discusses the computation of coordinate transformations equidistributing a monitor function based on exact solution information of a BVODE using MMODE0. Section 2.2 presents an algorithm which applies MT methods to adapt the solution of BVODEs, with experimental results given for the case where an arc-length based monitor function is used to govern adaptivity. In Section 2.3, this MT algorithm is applied to perform adaptation based on estimates of solution error. Section 2.4 describes the coupling of MT-based adaptation with an iterative refinement algorithm to give a simple, demonstrative example of an error control

MT algorithm for BVODEs.

For our experiments in this chapter we consider the BVODE [10]

$$y''(x) = \lambda^2(y(x) + \cos^2(\pi x)) + 2\pi^2 \cos(2\pi x), \tag{2.1}$$

with separated boundary conditions

$$y(0) = y(1) = 0. \tag{2.2}$$

For our purposes, it is useful to convert this BVODE to an equivalent system of first-order equations

$$\begin{aligned} y_1'(x) &= \lambda y_2(x), \\ y_2'(x) &= \lambda(y_1(x) + \cos(\pi x)) + \frac{2}{\lambda}\pi^2 \cos(2\pi x), \\ y_1(0) &= y_1(1) = 0, \end{aligned} \tag{2.3}$$

where $y_1(x) := y(x)$ and $y_2(x) := y'(x)$. The differential equation (2.3) has the exact solution

$$\begin{aligned} y_1(x) &= e^{\frac{\lambda(x-1)}{1+e^{-\lambda}}} - \cos^2(\pi x), \\ y_2(x) &= \frac{e^{\lambda(x-1)} - e^{-\lambda x}}{1 + e^{-\lambda}} + \frac{\pi}{\lambda}\sin(2\pi x), \end{aligned} \tag{2.4}$$

allowing us to directly evaluate the accuracy of the numerical methods being applied to this problem. Our reason for choosing this equation in our experiments is that increasing the value of the parameter λ increases the difficulty of the problem, allowing us to measure the performance of our algorithms on increasingly difficult problems. This is seen in Figure 2.1, where the exact solution for the first solution component of (2.3) with increasing λ values are plotted. Increasing λ in this problem increases the steepness of the boundary layers appearing in the solution, and consequently the difficulty of the problem.

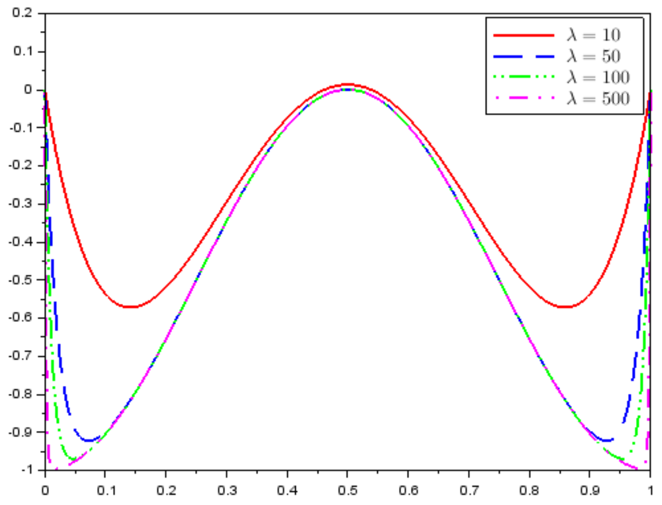


Figure 2.1: Solutions, $y_1(x)$, to BVODE (2.3), for several λ values.

In this chapter, we apply an MT algorithm to provide adaptivity within a computation that will yield a numerical solution to (2.3) under a variety of parameter and monitor function choices to examine the efficacy of the MT method as well as its suitability for implementation within an error control algorithm. The application of MT methods to BVODE problems is of little practical interest due to the many high-quality adaptive error control solvers available for this problem class which implement computationally inexpensive and robust direct r-adaptation algorithms [6, 12, 15]. However, analysis of this case will aid in the development of future algorithms for higher PDEs. As mentioned earlier, an example of a project that this case may help to inform is BACOL2D [9], since the tensor product B-spline Gaussian collocation algorithm implemented in this solver relies on the use of a rectangular spatial grid; the MT algorithms we consider in this thesis could potentially greatly benefit this code.

2.1 Computing the Coordinate Transformation

For the successful implementation of MT methods, it is necessary that a coordinate transformation is generated which can be used to transform a BVODE such that its solution is sufficiently smoothed in regions of difficult solution behaviour. This facilitates the accurate discretization of this transformed problem using a uniform mesh on Ω_c. To govern this smoothing, we require that the coordinate transformation satisfy the equidistribution principle, an essential component in most r-adaptation algorithms [4]. While an equidistributing coordinate transformation is desired, this can in general be difficult to obtain, as it requires a reasonably accurate solution to MMODE0. In MT methods, as well as in typical direct r-adaptive mesh refinement algorithms, it is recognized that for practical purposes the coordinate transformation, or in the discrete case, the mesh, need only be approximately equidistributing [4]. Further, for difficult problems, the monitor function will have regions where its value is large, corresponding to regions of difficult behaviour in the solution of the physical differential equation. These regions are interspersed with regions where the monitor function value is small, which correspond to regions where the solution to the physical differential equation is less challenging. While this kind of behaviour in the monitor function is essential for producing high-quality adaptivity and the coordinate transformation should ideally adapt to the regions where the monitor function is large, the result is that the solution to MMODE0, i.e., the coordinate transformation, can potentially be very non-smooth and thus difficult to compute with sufficient accuracy. This can potentially lead to a loss in monotonicity in the transformation, which will result in catastrophic errors when this transformation is used to transform the physical BVODE onto Ω_c. In general, the relationship between the smoothness of the monitor function and the solution to the MT equations is not very well understood; however, its important role in effective MT methods is well established.

Recall that the monitor function has been defined on the physical domain Ω_p. For this thesis, we require that, for use in MMODE0, the monitor function be mapped into the computational domain Ω_c. That is, in its original form we have the monitor function $M(x)$ defined on Ω_p, but we actually make use of $M(x(\xi))$, which represents a transformation of $M(x)$ between Ω_p and Ω_c.

A typical approach to address the difficulty of solving MT equations such as MMODE0 in MT algorithms is to apply a smoothing scheme such as a weighted moving average of discrete evaluations of the monitor function, thereby obtaining a spatially smoothed analog to the monitor function, resulting in an MMODE0 which is easier to solve [4]. In [4], monitor function smoothing is discussed in detail and the description of a continuous smoothing scheme based on the solution to the BVODE

$$\left(I - \gamma^{-2}\frac{d^2}{d\xi^2}\right)\bar{M}(\xi) = \hat{M}(\xi), \quad \bar{M}'(0) = \bar{M}'(1) = 0, \quad \gamma > 0, \quad \xi \in \Omega_c, \tag{2.5}$$

is given, where we recall that $\hat{M}(\xi) = M(x(\xi))$ is the mapping of the monitor function onto Ω_c. The solution to (2.5) is a smoothed monitor function $\bar{M}(x(\xi))$, obtained from the original monitor function $M(x(\xi))$, which is in this context interpreted as a function on the computational domain Ω_c. Discretizing (2.5) with a centred finite difference scheme, and taking a simple one-sided averaging at the boundary points, we obtain the local discrete smoothing scheme

$$\begin{aligned}
\tilde{M}_i &= \frac{1}{\gamma^2 h^2}\hat{M}_{i+1} + \left(1 - \frac{2}{\gamma^2 h^2}\right)\hat{M}_i + \frac{1}{\gamma^2 h^2}\hat{M}_{i-1}, \quad i = 2, ..., k, \\
\tilde{\hat{M}}_1 &= \frac{\hat{M}_1 + \hat{M}_2}{2}, \\
\tilde{\hat{M}}_{k+1} &= \frac{\hat{M}_k + \hat{M}_{k+1}}{2},
\end{aligned} \tag{2.6}$$

where $k+1$ is the number of discrete, uniformly spaced evaluations of the monitor function on Ω_c and h is the distance between these evaluations [4]. This scheme is typically applied for a number of iterations until the monitor function is sufficiently smooth to allow efficient and reasonably accurate computation of a solution to MMODE0. For our context, these discrete, smoothed monitor function evaluations are then interpolated using monotonic cubic splines [33] to produce a continuous computational monitor function for use in solving MMODE0, which will then lead to the generation of a coordinate transformation which equidistributes this smoothed analog of the monitor function.

The consequences of this smoothing are discussed further in this section. Choice of the parameter γ determines the intensity of the smoothing, with choices of γ such that $\gamma^2 h^2 \geq 2$ generating an averaging of the values [4]. The effect of smoothing using this scheme will be explored in detail in this chapter, and in our application of this method, we set $\gamma = \frac{\sqrt{2}}{h}$, which corresponds to choosing γ such that $\gamma^2 h^2 = 2$.

In this section, we experiment with the generation of approximately equidistributing coordinate transformations for use in an MT method; that is, the computation of coordinate transformations which are reasonably close to satisfying the equidistribution principle. We wish to understand the computational difficulties in efficiently computing a coordinate transformation, which corresponds to an approximately equidistributed mesh, through the solution of MMODE0, which we recall is defined on Ω_c and which has the form,

$$\frac{d}{d\xi}\left(M(x(\xi))\frac{d}{d\xi}x(\xi)\right) = 0, \tag{2.7}$$

with boundary conditions

$$x(0) = x_a, \quad x(1) = x_b. \tag{2.8}$$

To understand how to effectively solve MMODE0 and MT equations in general, we wish to assess the factors which affect the quality of the coordinate transformation and the difficulty of the process of solving (2.7). Factors such as the impact of monitor function smoothing on the efficiency of the computation of and the effectiveness of the coordinate transformation, and how the accuracy of the numerical solution to MMODE0 affects the quality of the coordinate transformation are investigated. These experiments are done using the *Scilab bvode* [11] error control BVODE solver discussed in Section 1.2 in order to compute the coordinate transformation to within desired levels of accuracy. Note that for the test problems, we have that $\Omega_c = \Omega_p = [0, 1]$; this fact is taken advantage of as a notational convenience.

When generating the coordinate transformation for an MT method, it is important that it is computed to a reasonable level of accuracy. Properties such as approximate equidistribution and monotonicity must be preserved in the solution as otherwise, transforming the physical BVODE using this transformation will either have no computational benefits or even have detrimental effects on the accuracy of the computed solution to the physical BVODE. The choice of monitor function plays an important role in generating an effective coordinate transformation. If the monitor function does not accurately reflect regions of solution difficulty, then the MT method will be ineffective. As mentioned previously, if the monitor function varies too rapidly then MMODE0 can become difficult to solve and possibly even fail to converge to a solution. Therefore, an effective method for generating the coordinate transformation must include a monitor function which balances accurately indicating areas of solution difficulty with being reasonably smooth.

For the experiments in this section we make use of the arc-length monitor function [4, 5], which for a BVODE system with n solution components has the form,

$$\hat{M}(\xi) = M(x(\xi)) = \sqrt{1 + \sum_{i=1}^{n} \left(\frac{d}{dx} y_i(x(\xi)) \right)^2}. \tag{2.9}$$

This monitor function is a popular choice in MT methods, due to the fact that regions where the solution to a BVODE is varying rapidly, and where the arc-length is large, are difficult to discretize accurately, potentially leading to an approximate solution with large errors in these regions [4]. Therefore, adapting the computation by smoothing the physical equation in regions where the arc-length is large will allow it to be more accurately discretized using a uniform mesh in Ω_c. For this section of this chapter, the arc-length monitor function is based on the exact solution (2.4), though in more general contexts this will, of course, be unavailable. For problem (2.3), the exact arc-length monitor function is plotted for various values of the parameter λ in Figure 2.2. We see here that as λ is increased, the value of the arc-length monitor function becomes very large towards edges of the domain, which corresponds to the steep layer regions present in the solution, seen previously in

Figure 2.1. In the case where $\lambda = 50$, several iterations of the discrete spatial smoothing scheme (2.6) were performed and the resultant smoothed monitor functions are shown in Figure 2.3. This demonstrates the effects of this smoothing algorithm, with the resultant smoothed monitor functions varying far more gradually while largely retaining qualitatively similar behaviour, compared to the original monitor function.

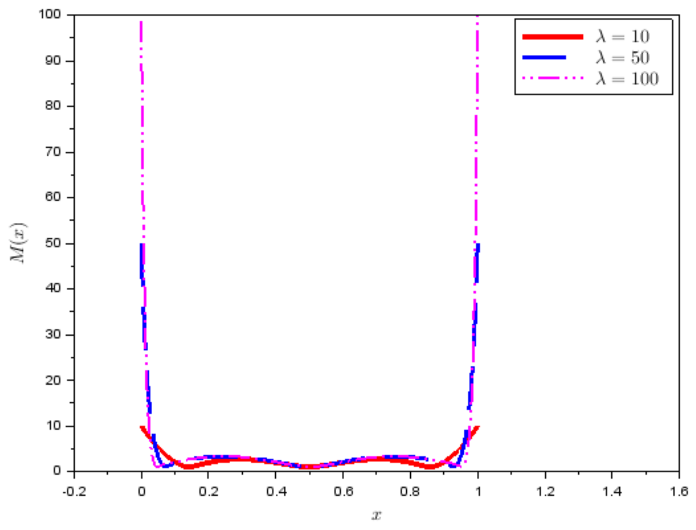


Figure 2.2: Arc-length monitor function for BVODE (2.3) for several choices of the λ parameter.

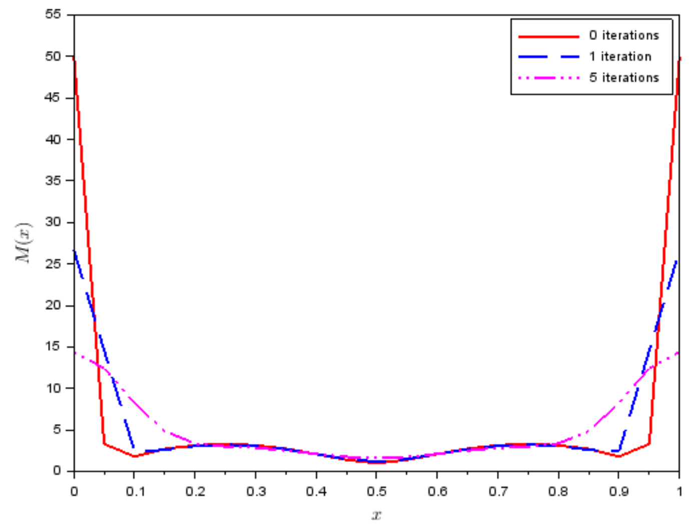


Figure 2.3: Smoothing of the exact arc-length monitor function for (2.3) with $\lambda = 50$.

It is important that the computation of a solution to an MT equation balances the efficiency of the computation and quality of the equidistribution of the solution. Smoothing of the monitor function may assist with the efficiency and ease of the solving MMODE0, but it results in a coordinate transformation which instead of equidistributing the monitor function of interest equidistributes only its smoothed analog, which as Figure 2.3 shows, can differ substantially from the original.

In order to assess the quality of the coordinate transformation $x(\xi)$ obtained as the solution to MMODE0, we wish to examine how well $M(x)$, represented now in Ω_p, is equidistributed by this transformation. In particular, we are interested in how well $x(\xi)$ satisfies the relation

$$\int_{x(\xi_1)}^{x(\xi_2)} M(x)dx = ... = \int_{x(\xi_N)}^{x(\xi_{N+1})} M(x)dx = \frac{\sigma}{N}, \tag{2.10}$$

where $\{\xi_i\}_{i=1}^{N+1}$ is a uniform computational mesh of N subintervals on Ω_c and

$$\sigma = \int_{x_a}^{x_b} M(x)dx. \tag{2.11}$$

That is, we measure the quality of $x(\xi)$ in terms of how well that meshes generated by evaluating it at the points of a uniform computational mesh equidistribute the exact arc-length monitor function. This can be tested by approximating each integral in (2.10) and (2.10) numerically to high accuracy and then measuring the relative departure from equidistribution, E_{eq}, given by

$$E_{eq}(x) = \max_{i=1,\ldots,N} \left\{ \left| \frac{\int_{x(\xi_i)}^{x(\xi_{i+1})} M(x(\xi))dx - \frac{\sigma}{N}}{\frac{\sigma}{N}} \right| \right\}. \tag{2.12}$$

That is, E_{eq} is the maximum relative departure from equidistribution over all subintervals of the mesh $x = \{x(\xi_i)\}_{i=1}^{N+1}$, generated as the coordinate transformation, $x(\xi)$, obtained from the solution to MMODE0. Note that a perfectly equidistributing coordinate transformation would have $E_{eq}(x) = 0$.

We now begin testing the approximately equidistributing coordinate transformations obtained by solving MMODE0. For each of these tests, we solve for $x(\xi)$ using *bvode* while varying the number of iterations of the monitor function smoothing scheme. Note that (2.7) can be written as the second order ODE

$$\left(\frac{d}{d\xi}M(x(\xi))\right)\left(\frac{d}{d\xi}(x(\xi))^2\right) + M(x(\xi))\left(\frac{d^2}{d\xi^2}x(\xi)\right) = 0, \tag{2.13}$$

For use with *bvode*, MMODE0 is converted into the equivalent first order system of equations (although *bvode* does have the option for the direct treatment of MMODE0),

$$\begin{aligned}
\frac{d}{d\xi}u_1(\xi) &= u_2(\xi),\\
\frac{d}{d\xi}u_2(\xi) &= -\frac{d}{d\xi}M(x(\xi))\frac{u_2(\xi)}{M(x(\xi))},\\
u_1(0) &= x_a, \quad u_1(1) = x_b.
\end{aligned} \tag{2.14}$$

where $u_1(\xi) := x(\xi)$ and $u_2(\xi) := \frac{d}{dx}u_1(\xi)$.

Up to this point, we have written our monitor function to show explicit dependence on $x(\xi)$, namely, $M(x(\xi))$; however, $x(\xi)$ is, of course, unavailable when the monitor function is being constructed. While this is seemingly inconsequential when viewing the monitor function as simply a function of the physical domain $M(x)$, note that MMODE0 is written such that it has a non-linear dependence on $M(x(\xi))$, since $x(\xi)$ is the solution to this equation. To reformulate our approach so as to compensate for the use of the smoothed computational monitor function in the way previously described, we consider the monitor function to be an implicit function of some coordinate transformation $\widetilde{x}(\xi)$, which is assumed to have been previously obtained. Then we consider the analog of this monitor function on Ω_c,

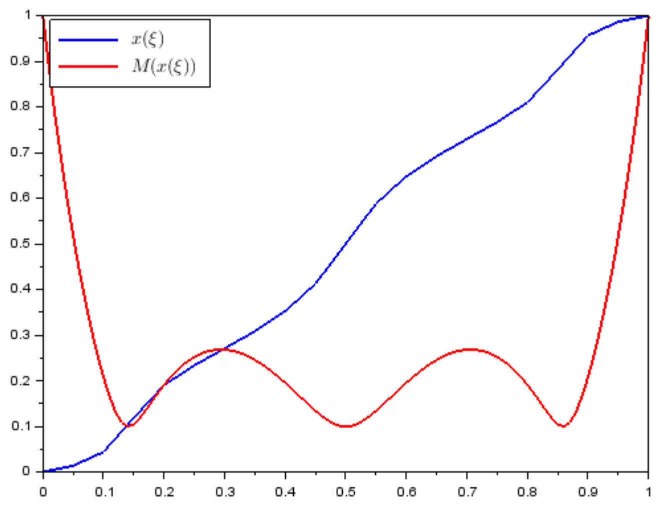
$$\hat{M}(\xi) := M(\widetilde{x}(\xi)). \tag{2.15}$$

Rewriting MMODE0 in terms of this monitor function, we have

$$\begin{aligned} \frac{d}{d\xi}\left(M(\widetilde{x}(\xi))\frac{d}{d\xi}x(\xi)\right) &= 0, \\ \frac{d}{d\xi}\left(\hat{M}(\xi)\frac{d}{d\xi}x(\xi)\right) &= 0, \end{aligned} \tag{2.16}$$

Where $\hat{M}(\xi)$ is not $M(\widetilde{x}(\xi))$. Of course, this formulation is a departure from the previously stated theory, particularly when $\widetilde{x}(\xi)$ and the equidistributing coordinate transformation $x(\xi)$ are dissimilar. This approach is analogous to those sometimes used within alternating procedures for the time-dependent PDE case [4]. Note that in practice, we will initially have $\widetilde{x}(\xi) = \xi$, which we refer to as the uniform transformation and a substantial number of monitor function smoothing iterations as it corresponds to the use of a uniform mesh in the discrete case. Since the uniform transformation will typically be quite far from the equidistributing coordinate transformation for a particular monitor function, iterative approaches can be used to make the implementation more faithful to the theory by first solving MMODE0 with the uniform transformation and using the resultant coordinate transformation to again, solve MMODE0. However, the use of this kind of approach adds significant cost to an already expensive algorithm, so we examine the efficacy of this approach when simply using the uniform transformation $x(\xi) = \xi$. Later we will experiment with the use of iterative procedures to address this issue.

For (2.3) with $\lambda = 10$, Figure 2.4 compares the coordinate transformation generated as the solution to MMODE0, $x(\xi)$, using *bvode* with a mesh, $\widetilde{x}$, of 20 subintervals, obtained using de Boor's algorithm, $\bar{x}$. These are plotted against the normalized arclength monitor function, i.e., $\frac{M(x)}{\max\{M(x)\}}$, restricting its values to between 0 and 1. Both the coordinate transformation and the mesh generated using de Boors algorithm were obtained using the exact arc-length monitor function, and an error tolerance of 10^{-6} supplied to *bvode*. No smoothing of the monitor function is done in this example. Note that in order to compare the coordinate transformation $x(\xi)$ with the mesh $\bar{x}$ in this way, the points $\{(\xi_i, \bar{x}_i)\}_{i=1}^{N+1}$ are plotted, where for each i, ξ_i corresponds to the ith mesh point of a uniform mesh defined on Ω_c and $\bar{x}_i$ is the ith mesh point of $\bar{x}$.



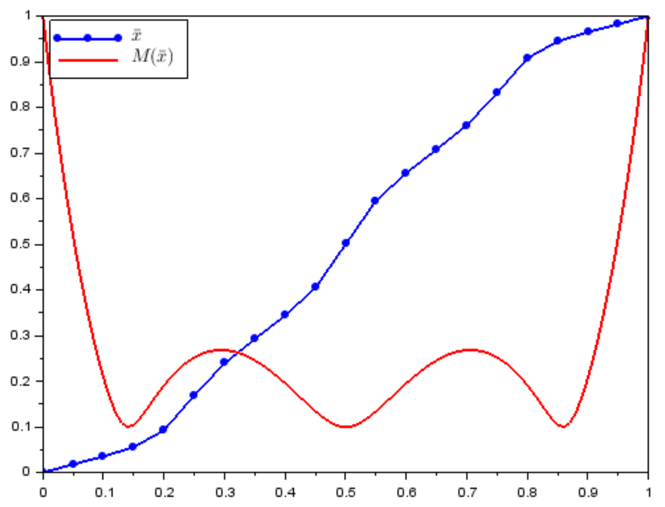


Figure 2.4: In the top plot we show the coordinate transformation $x(\xi)$ obtained from the numerical solution to MMODE0 (2.14) along with a normalized plot of the exact arc-length monitor function for (2.3) with $\lambda = 10$. In the bottom plot we show the mesh obtained from de Boor's algorithm based on the same monitor function as in the top plot.

Note that in these plots, locations where $x(\xi)$ or $\bar{x}$ are flatter correspond to regions where, in the continuous setting, the independent variable on Ω_c is stretched so that the solution to the BVODE will be easier to compute. In the discrete setting, these flatter regions correspond to locations where more mesh points are clustered. From this we can see that in each of these cases adaptation is being done in response to regions where the monitor function is large. Note that the de Boor mesh, $\bar{x}$, concentrates more mesh points at the boundaries, where $M(x)$ is largest. The de Boor mesh, $\bar{x}$, is closer to equidistribution than the mesh generated by $x(\xi)$, with $E_{eq}(\bar{x}) = 0.21$, $E_{eq}(x(\xi)) = 0.83$. As a reference, the E_{eq} value for a uniform mesh on this problem is 1.84. Figure 2.5 shows the values of the per-subinterval integrals of the monitor function for $x(\xi)$, plotted against the target value shown in green for each subinterval. Large values at the boundaries indicate that more mesh points should have been concentrated at these locations. Additionally, we see regions where the values are much lower than required, indicating that too many points have been placed there.

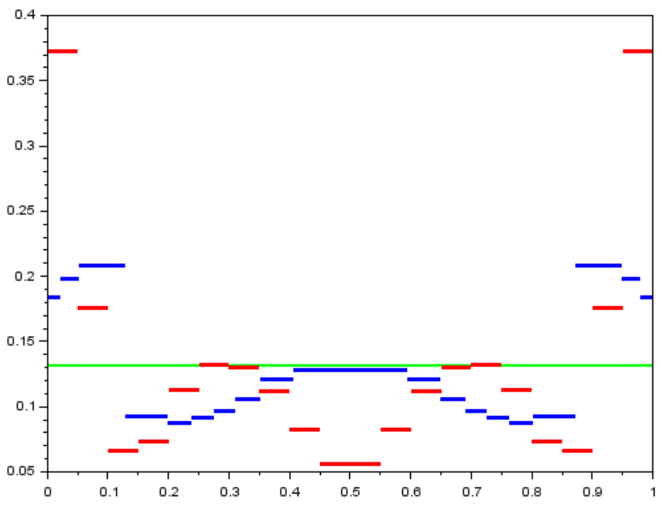


Figure 2.5: Per-subinterval departure from equidistribution for MMODE0 generated mesh (blue) and uniform mesh (red) in Figure 2.4. Plotted against the target per-subinterval value, shown in green.

On this same problem, another coordinate transformation was generated by MMODE0 after one iteration of monitor function smoothing. In this test case, the cost of the computation to solve MMODE0 was substantially reduced; the first computation that used no smoothing took almost

3 times as long to complete but we see from Table 2.1 that both computations give comparable results. The use of a smoothed monitor function results in MMODE0 having a smoother solution which reduces the difficulty of solving MMODE0. Figure 2.6 demonstrates this effect by plotting the mesh in Ω_p that corresponds to the coordinate transformation, for (2.3) with $\lambda = 50$. A mesh generated using de Boor's algorithm is compared with the MMODE0 generated mesh, where when solving MMODE0, a monitor function with 5 iterations of the smoothing was used, and de Boor's algorithm was supplied with the exact monitor function (2.9). From Figure 2.6, we see that the coordinate transformation resulting from the solution of the smoothed MMODE0 has much less variation in its behaviour.

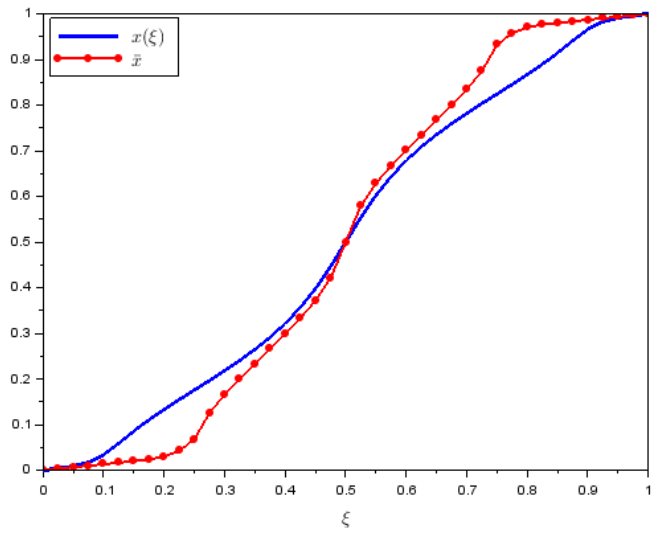


Figure 2.6: Comparison of smoothed MMODE0 solution $x(\xi)$ with the de Boor generated mesh of $N = 20$ subintervals. $x(\xi)$ is based on the interpolated exact arc-length monitor function for (2.3) with $\lambda = 50$, with 5 iterations of (2.6) applied; $\bar{x}$ was obtained using the exact arc-length monitor function. $E_{eq}(x(\xi)) = 1.31$ and $E_{eq}(\bar{x}) = 0.54$.

The results showing the relationship between the choice in arc-length monitor function (as governed by the problem parameter λ), the number of iterations of smoothing scheme (2.6), and E_{eq} for (2.3) are summarized in Table 2.1. Note that in these test cases *bvode* uses a self-generated initial mesh and is given the uniform transformation $x(\xi) = \xi$ as the initial guess for the solution to MMODE0. The transformation of the monitor function $M(x)$ defined on Ω_p to the transformed monitor function $\hat{M}(\xi)$ on Ω_c is obtained using the uniform transformation. Table 2.1 also includes corresponding results of the mesh obtained through the application of de Boor's algorithm using $M(x)$, and results for a uniform mesh.

Interpreting the data from this table, we can make the following observations:

λ	Smoothing Iterations						
	1	5	10	50	100	de Boor's	Uniform
10	0.630134	0.560363	0.850941	1.532508	1.748086	0.210505	1.835616
50	1.445442	1.208950	1.769777	3.182369	3.455642	0.866034	3.536085
100	1.766506	1.396611	2.095343	3.604891	3.706865	1.463078	3.733072
500	ERR	1.560175	2.542617	5.810137	6.034042	4.026082	6.161724

Table 2.1: E_{eq} (2.3) with for varying λ choices and number of iterations of smoothing scheme (2.6). E_{eq} is computed with $N = 20$.

- As the monitor function is smoothed, the resultant coordinate transformation tends to move further from equidistribution, as expected, with the exception to this being in the case where 5 iterations of the smoothing scheme have been applied, in which case the quality of the coordinate transformation increases. This unexpected behaviour in this case is likely a consequence of the use of the uniform transformation to define the computational monitor function.

- When λ is increased E_{eq} tends to increase, as expected.

- When λ is large and few iterations of monitor function smoothing are applied, *bvode* is unable to solve MMODE0 to the requested tolerance. See the case $\lambda = 500$ with 1 smoothing iteration. We see that the uniform mesh always has larger E_{eq} values than the adapted meshes. We also see that the mesh generated by de Boor's algorithm performs, in general, better than the mesh generated as the solution to MMODE0 for the easier test problems. For the more difficult problems, we see that the MMODE0 generated mesh is more effective.

The first three of these observations are expected since this smoothing process means that a perturbed monitor function is used to generate the coordinate transformation and thus taking it away from its original form. Therefore, the original monitor function will not be equidistributed particularly well by the resultant $x(\xi)$ transformation. Further, as the problem becomes more difficult, and hence the monitor function has higher variation, computation of the solution to MMODE0 that leads to exact equidistribution becomes very difficult to obtain, so difficult that even a high-quality solver such as *bvode* can fail to produce it.

While we see that the coordinate transformation becomes further from equidistribution as the number of smoothing iterations is increased, this smoothing comes at a substantial benefit in terms of execution time. Table 2.2 gives the CPU times for each of the tests reported in the previous table, with each time reported as the median of 5 runs. Here we see that monitor function smoothing provides a large performance benefit when solving MMODE0. From the combined results of Tables 2.1 and 2.2, we see that the performance gains in terms of CPU time come at the price of producing a less well equidistributed coordinate transformation. However, one of the principles of MT methods is that the coordinate transformation need not be computed to particularly high levels of accuracy, so this trade-off may, in many cases, be worth it. The effects of this trade-off in terms of quality of equidistribution of the coordinate transformation and number of mesh points required in the uniform meshes used in Ω_c in order to obtain the desired accuracy are discussed in later sections. From each of these tables, we can also see that in general, solutions generated using the monitor function smoothing to determine the degree of smoothness balance both efficiency and closeness to equidistribution. This motivates further use of and experimentation with mesh smoothing in this chapter.

λ	Smoothing Iterations				
	1	5	10	50	100
10	2.515625	1.250000	1.093750	0.234375	0.218750
50	8.937500	1.140625	1.187500	0.671875	0.250000
100	19.359375	2.062500	1.078125	0.484375	0.093750
500	ERR	3.390625	2.078125	1.156250	1.078125

Table 2.2: CPU time when solving MMODE0 for different λ choices and number of smoothing iterations with *bvode*.

As mentioned previously in this section, the use of a computational monitor function based on the uniform transformation causes the method to stray from the mathematical formulation. This is due to the fact that the uniform transformation used to generate the computational monitor function used in MMODE0 can differ greatly from an equidistributing coordinate transformation. To see the effects of this, the generation of the coordinate transformation was iterated, with the monitor function being regenerated on each step using the previously computed coordinate transformation. The results of this test are summarized in Table 2.3, with 5 iterations of the monitor function smoothing scheme applied for each coordinate transformation. Here we see that this kind of iteration results in substantial improvements in how well equidistributing the coordinate transformation becomes. For this reason, we will implement this kind of iteration into later sections, where solutions to MMODE0 are applied in an MT algorithm.

λ	Iterations				
	1	2	3	4	de Boor
10	0.698340	0.404200	0.344729	0.374402	0.108657
50	1.308735	0.840083	0.591653	0.563335	0.536279
100	1.577589	1.101601	0.583774	0.557757	1.031814
500	1.702371	1.194756	0.712521	0.776057	3.880879

Table 2.3: E_{eq} values from iterative generation of the coordinate transformation, 5 iterations of monitor function smoothing, $N = 40$.

It is also worth mentioning that in our testing we did not find any significant cases where the solution to MMODE0 lost monotonicity, except in certain cases when no smoothing was applied to the monitor function. In most cases, unless $x(\xi)$ is extremely difficult to compute, the solution to MMODE0 will preserve monotonicity. The importance of preserving monotonicity in the solution to MT equations has motivated the rigorous analysis of the discretization methods used to solve them to ensure that monotonicity will be preserved, such as the analysis performed in [34]. A high-quality MT software package would be expected to have theoretical and practical guarantees that the coordinate transformation preservers monotonicity for all possible values of the monitor function upon which it is based. In our experimentation, we also observed that the accuracy to which the coordinate transformation is computed does not, in general, have a significant impact on the quality of the equidistribution. Table 2.4 shows the E_{eq} values in the $\lambda = 100$ case for varying tolerances used with *bvode* to solve MMODE0 computed for a mesh of 20 subintervals. We see from this that fairly low accuracy computations can be used to generate the coordinate transformation, so long as it retains monotonicity. This fact is often exploited in software implementing MT methods. Lower order, more computationally efficient numerical methods are often used to solve for the MT equation, and more rapidly converging and robust methods are used to solve the transformed physical differential equation. This can be seen in moving mesh codes such as MOVCOL, where PDEs are discretized using a cubic Hermite collocation algorithm and the MT equation is discretized using a simple three-point finite difference method [8].

TOL	Smoothing Iterations				
	1	5	10	50	100
10^{-2}	1.767862	1.397828	2.095480	3.604891	3.706865
10^{-4}	1.766504	1.396611	2.095480	3.604891	3.706865
10^{-6}	1.766506	1.396611	2.095343	3.604891	3.706865

Table 2.4: Effect of *bvode* error tolerance choice on the quality on E_{eq}, $\lambda = 100$, $N = 20$.

In this section, we have described the problem and computational difficulties of computing MMODE0. The effects of monitor function smoothness, choice of monitor function, and other factors were measured in order to understand how to practically compute an effective coordinate transformation for use in an MT method for BVODEs. This helps us to understand one of the core components of an MT method, which will aid our exploration in further sections, where we begin to apply MT methods to adapt the solution to the physical BVODE.

2.2 Moving Transformation Approach for BVODEs Using an Arc-Length Monitor Function

In Section 2.1, we developed an understanding of how MMODE0 can be solved to generate approximately equidistributed coordinate transformations when using the exact arc-length monitor function. We now wish to apply the MT method, with coordinate transformations generated as described in the previous section, to solve a BVODE. The coordinate transformation is applied to a BVODE on a physical domain to transform it to a BVODE in the computational domain with a smoother solution, allowing it to be effectively solved using a uniform mesh. The effects of this approach on the accuracy of the computed solution to the physical BVODE will be measured experimentally, along with other factors such as the effect of monitor function smoothing on the quality of the computed solution. As a reminder, the MT method transforms a BVODE $y'(x) = f(x, y(x))$, where $x \in \Omega_p$, by coupling it with the derivatives of the coordinate transformation, generating the transformed BVODE which has a smoothed solution on the computational domain; the transformed BVODE is

$$\frac{d}{d\xi}\hat{\underline{y}}(\xi) = f(x(\xi), \hat{\underline{y}}(\xi))\frac{d}{d\xi}x(\xi), \tag{2.17}$$

where $x(\xi)$, the coordinate transformation, is the solution to MMODE0 (2.14), $\hat{\underline{y}}(\xi) := \underline{y}(x(\xi))$, and $\xi \in \Omega_c$.

The monitor function used in this section will be the arc-length monitor function using approximate solution information, as opposed to the exact arc-length monitor used in Section 2.1. When implementing a monitor function which depends on the approximate solution such as the arc-length monitor or a monitor function based on solution error estimates, the monitor function is initially unknown, as there is no solution information with which to define it. In these cases, we first make use of the uniform transformation $x(\xi) = \xi$, which corresponds to the constant monitor function $M(x) = 1$ and reduces the transformed BVODE (2.17) to the original BVODE defined on the physical domain. Solving the untransformed BVODE with a uniform mesh on Ω_p generates an initial, inaccurate approximate solution. This initial solution is used to generate the monitor function, using either information about the approximate solution itself such as its arc-length or an error estimate for the approximate solution. The coordinate transformation can then be obtained as the solution of MMODE0 using this monitor function to govern the adaptivity. This then allows the coupled system (2.17) to be transformed into the computational domain Ω_c with some degree of smoothing, where it can be solved accurately using a uniform mesh.

As discussed in Section 2.1, when implementing an interpolated monitor function, we make use of its transformed analog on Ω_c, which must be obtained using a previous transformation. To obtain a reasonably well-suited transformed monitor function, we iteratively solve MMODE0 three times, which as we saw in the previous section improves how well the resultant coordinate transformation equidistributes $M(x)$, the original monitor function on Ω_p (see Table 2.3). We also note that the initial solution approximation used to obtain the monitor function, and the initial uniform mesh both provide natural choices for initial guesses for the BVODE solver, helping to accelerate the convergence of the algorithm. This procedure is summarized in Algorithm 3. Generation of an initial approximate solution with which to drive adaptivity is the basis for almost every adaptation algorithm for differential equations and is central in error control solvers such as BACOLI and COLNEW.

Algorithm 3: MT BVODE Basic Algorithm

```
1  function MTBVODE ($N$, $x_a$, $x_b$, $BVODE$, $MMODE$);
   Input  : Number of subintervals $N$; left boundary $x_a$; right boundary $x_b$; problem definition
            $BVODE$; moving mesh ODE $MMODE$
   Output: Adapted solution to $BVODE$ $\underline{\mathrm{Y}}$(x)
2  // Generate uniform computational mesh
3  $\xi := linspace(x_a, x_b, N+1)$
4  // Solve untransformed $BVODE$ on uniform mesh with initial guess $\underline{0}$
5  $\underline{\hat{Y}}(\xi) := Y\_SOL(\xi, BVODE, \underline{0})$
6  // Generate the initial monitor function
7  $\hat{M}(\xi) :=$ GenerateMonitor($\xi$, $\underline{\hat{Y}}$)
8  // Solve for coordinate transformation using the monitor function and initial guess $\xi$
9  $x(\xi) := X\_SOL(\xi, MMODE(\hat{M}(\xi)), \xi)$
10 // Solve the transformed $BVODE$ using the initial solution as the initial guess
11 $\underline{\hat{Y}}(\xi) := Y\_SOL(\xi, BVODE(x(\xi)), \underline{\hat{Y}})$
12 // Interpolate the solution in $\Omega_c$ to generate the solution on $\Omega_p$.
13 $\underline{Y}(x) :=$ interp($x(\xi), \underline{\hat{Y}}(\xi)$)
14 return $\underline{Y}(x)$
```

For the experiments performed in this section, we make use of the Midpoint scheme, described in Chapter 1, for the solution to MMODE0 as well as the transformed BVODE system. The use of a discretization for MMODE0 which is either as accurate or less accurate than that used on the physical PDE is well-justified, as the primary purpose of these methods is to accurately solve the physical BVODE rather than the MMODE. Further, as we saw in the previous section, the MMODE0 will typically not need to be solved particularly accurately to produce a reasonably effective transformation. In MT methods, it is standard to implement numerical methods which generate discrete solution approximations. However, standard adaptive error control solvers often rely on the use of continuous solution approximations for their error control algorithms, with these continuous solution approximations being particularly useful when the number of mesh points must be changed dynamically. For the Midpoint scheme, it is simple to build off of the discrete solution approximations this discretization generates to obtain a continuous solution approximation with a consistent order of global accuracy. This can be done by interpolating the $\mathcal{O}(h^2)$ accurate discrete solution information generated by the Midpoint scheme using Hermite cubic splines [13], which have $\mathcal{O}(h^4)$ interpolation error, to produce a continuous solution which is globally $\mathcal{O}(h^2)$ accurate. To enforce monotonicity of the coordinate transformation, the discrete evaluations of the solution to MMODE0 generated by the Midpoint scheme are interpolated using monotonicity preserving cubic splines [33], which have interpolation error of $\mathcal{O}(h^3)$, meaning the coordinate transformation will be globally $\mathcal{O}(h^2)$ accurate. To observe the effects on solution error that the transformation of the physical BVODE induces, we compare these results with those obtained by computing the solution to the physical BVODE by discretizing it directly using the Midpoint scheme on the non-uniform mesh derived from $x(\xi)$, $\{x(\xi_i)\}_{i=1}^{N+1}$ (traditional r-adaptivity). In this way, we can measure the performance of MT methods in comparison to the more traditional direct r-adaptation approach.

In an MT method, it is vital that the transformed differential equation has a solution which is smoothed in regions as indicated by the monitor function. When applying the arc-length monitor function in the MT method to (2.3) with parameter $\lambda = 100$, we apply the Midpoint scheme on a uniform mesh of $N = 20$ subintervals on Ω_c to solve both the transformed physical BVODE and MMODE0. The solutions to the physical BVODE on the physical domain and to the transformed physical BVODE in the computational domain are plotted in Figure 2.7. The approximate solution to the physical BVODE in the original domain Ω_p is obtained by transforming the computed solution of the transformed physical BVODE on Ω_c back to Ω_p by interpolating at the points $\{\hat{\underline{Y}}(\xi_i)\}_{i=1}^{N+1} = \{\underline{Y}(x(\xi_i))\}_{i=1}^{N+1}$, where $\underline{Y}(x(\xi))$ is the approximate solution on Ω_p and $\hat{\underline{Y}}(\xi)$ is its analog on Ω_c. Here we see that the regions near the boundaries of the highest solution difficulty become smoothed when the BVODE is transformed to Ω_c. This smoothing effect has a substantial impact in terms of solution accuracy, which is demonstrated in Figures 2.8, 2.9, and 2.10, where for several λ values, solutions generated by directly solving the untransformed BVODE on a uniform mesh, and by using the MT method to smooth the BVODE are plotted.

Comparing the results in Figures 2.8 - 2.10 with the exact solutions given previously in Figure 2.1, we see the potential efficacy of the MT method in improving the accuracy of solutions generated using standard numerical methods for BVODEs, with the MT generated solutions being significantly closer

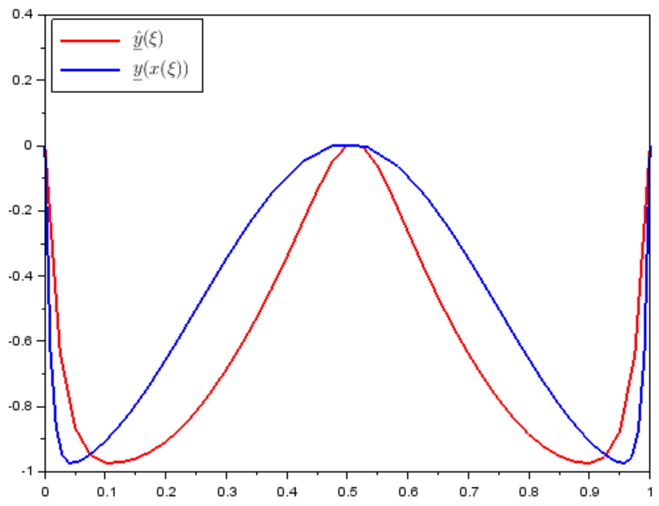


Figure 2.7: Solution to (2.3) with $\lambda = 100$, smoothed using the arc-length monitor function in the MT method with the Midpoint scheme used for spatial discretization, $N = 20$. $\hat{y}(x(\xi))$ is the transformed computed solution on $\Omega_c = [0, 1]$. $y(x(\xi))$ is the transformed computed solution on $\Omega_p = [0, 1]$.

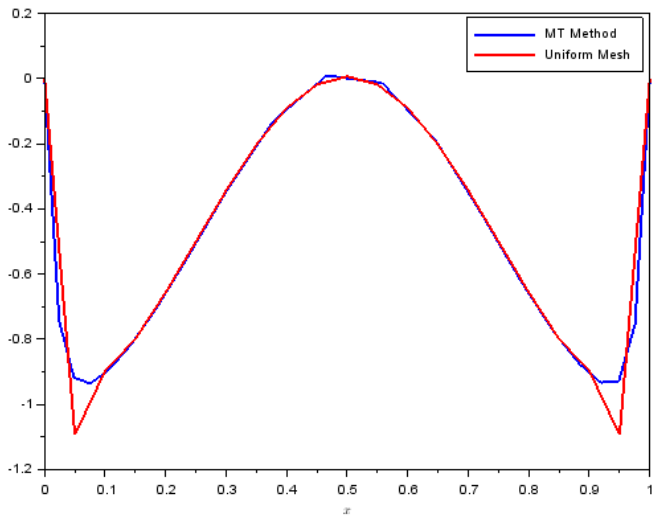


Figure 2.8: Solution to (2.3) with $\lambda = 50$ computed using the MT algorithm and then transformed onto Ω_p. Plotted against solution computed directly on a uniform mesh on Ω_p; $N = 20$.

to the exact solution. However, there are several factors to consider before an algorithm using MT adaptivity could be implemented in an error control framework. In order for a solution discretized on a uniform mesh to have reasonable accuracy it must be effectively smoothed by the MT method; otherwise, an extremely fine mesh will likely have to be used, potentially a substantial inefficiency.

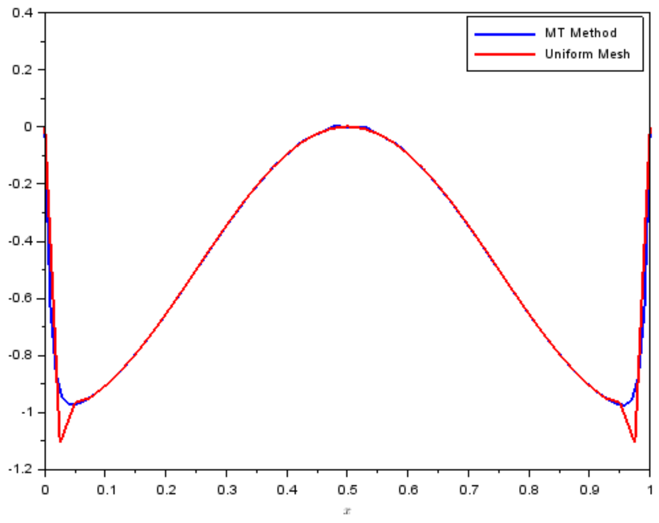


Figure 2.9: Solution to (2.3) with $\lambda = 100$ computed using the MT algorithm and then transformed onto Ω_p. Plotted against solution computed directly on a uniform mesh on Ω_p; $N = 40$.

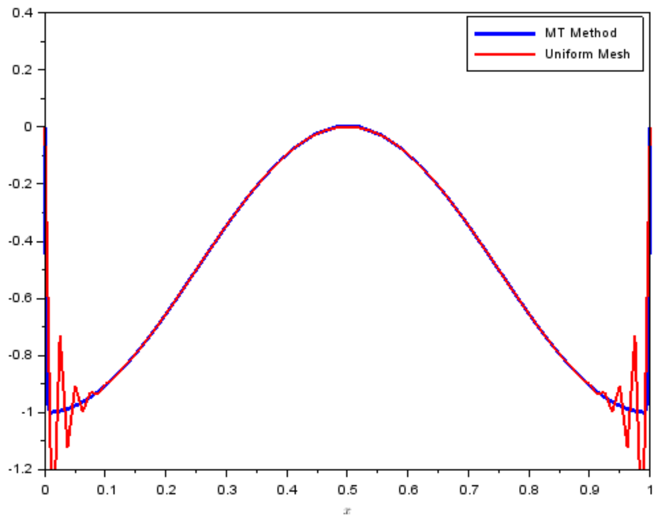


Figure 2.10: Solution to (2.3) with $\lambda = 500$ computed using the MT algorithm and then transformed onto Ω_p. Plotted against solution computed directly on a uniform mesh on Ω_p; $N = 80$.

Therefore, the MT method must provide a reasonable level of adaptivity.

Another factor to consider is how well MT methods perform relative to existing adaptation methods in terms of their ability to adapt the solution to a BVODE. If the MT method can't perform similarly to standard r-adaptation methods then other adaptation approaches should be considered. While clearly there exist better alternatives to MT methods for BVODEs and 1D

PDEs, as mentioned earlier, the MT approach becomes valuable when considering its use for higher dimensional problems, where direct adaptive methods are not applicable. Therefore we must see how well MT adaptation with the transformation of the solution to Ω_c compares with the standard approach of discretizing on a non-uniform mesh in Ω_p.

Another important factor is whether the addition of more mesh points to the uniform mesh on the computational domain can always result in an improvement in solution accuracy. This is an essential assumption in the standard hr-adaptation approaches implemented in software such as BACOLI and COLNEW. As discussed in the previous section, monitor function smoothing and the resultant smoothness of the coordinate transformation has a large impact on computational efficiency and how accurately MMODE0 can be solved on a uniform mesh. To understand each of these factors, a series of tests were run, comparing the error in the solution generated with the MT algorithm, as well as solutions obtained by directly discretizing in the physical domain, both on a uniform mesh and on the mesh $\{x(\xi_i)\}_{i=1}^{N+1}$, obtained by mapping the uniformly spaced computational mesh points onto the physical domain (traditional r-adaptivity). The first of these comparisons is considered in Figures 2.8 - 2.10. The results comparing the errors of the MT generated solution and the uniform mesh generated solution are summarized in Table 2.5. Note that the error results are reported as the maximum error at each of the transformed mesh points, with the error in the MT case being reported as $\max_{i=1,\ldots,N+1}\{||\underline{Y}(x(\xi_i)) - \underline{y}(x(\xi_i))||_\infty\}$, where $\underline{y}$ is the exact solution to (2.3), $x(\xi)$ is the coordinate transformation and $\underline{Y}$ is the approximate solution to the BVODE. The results for direct discretization on the MMODE-generated mesh are given in Table 2.6, with the comparison to the uniform mesh case for reference.

Table 2.5 provides some important insights into how an MT method can function in practice. Firstly, we note that in almost all of the test cases the MT method outperforms (i) the uniform mesh generated solution and (ii) performs comparably to the direct discretization approach as seen in Table 2.6. An exception to the improved performance comes in the $\lambda = 10$ case. For simple problems such as this, it is often sufficient to solve the problem directly using a uniform mesh, so improved performance in this case was not expected. Comparing these results with those in

N	Smoothing Iterations					
	1	5	10	50	100	**Uniform**
$\lambda = 10$						
20	0.010879	0.012339	0.011561	0.009239	0.010205	**0.010348**
40	0.002540	0.002634	0.002776	0.002635	0.002137	**0.002544**
80	0.000629	0.000626	0.000631	0.000693	0.000705	**0.000634**
160	0.000157	0.000156	0.000156	0.000157	0.000163	**0.000158**
$\lambda = 50$						
20	0.033765	0.035798	0.050848	0.166051	0.195989	**0.199678**
40	0.031909	0.009011	0.007078	0.013709	0.026284	**0.056882**
80	0.005111	0.003297	0.002893	0.001789	0.002383	**0.012398**
160	0.001419	0.001156	0.000958	0.000689	0.000540	**0.003078**
$\lambda = 100$						
20	0.072543	0.113251	0.159457	0.397821	0.440576	**0.443862**
40	0.150093	0.017385	0.022484	0.054341	0.102334	**0.194886**
80	0.021870	0.004553	0.003809	0.006200	0.008488	**0.056029**
160	0.005127	0.002958	0.001934	0.001000	0.001251	**0.012198**
$\lambda = 500$						
20	27.54271	0.626856	0.696331	0.860343	0.875462	**0.875695**
40	0.810914	0.253738	0.302908	0.481678	0.610645	**0.726789**
80	0.052149	0.062306	0.077061	0.126635	0.162068	**0.517665**
160	0.014516	0.011953	0.013198	0.020564	0.026488	**0.263567**

Table 2.5: Error of the numerical solution computed by the MT method using the arc-length monitor function with various degrees of monitor function smoothing and spatial discretization is done using the Midpoint scheme. Applied to (2.3) for several λ values and several values for N.

Table 2.6, where the BVODE is directly discretized on the non-uniform mesh $\{x(\xi_i)\}_{i=1}^{N+1}$, we see comparable performance results, with the error in both cases having a similar dependence on the smoothness of the monitor function.

Further examining the relationship between monitor function smoothing and solution accuracy, we see that in many cases, when the monitor function is smoothed for many iterations, the error we obtain is close to the error in the uniform mesh case. This is expected from the results in the previous section, as smoothing results in a transformation which is increasingly far from equidistribution. When the monitor function is smoothed for too many iterations, it can become close to constant, resulting in a nearly uniform transformation, and consequently poor adaptivity when applying the MT method. In many cases, we also see that some amount of smoothing of the monitor function results in improvements in the quality of the solution. From this, we see that monitor function smoothing and obtaining a coordinate transformation with effective equidistribution must be balanced in order to have effective computations. In Table 2.5, we see that in all test cases increasing

N	Smoothing Iterations					
	1	5	10	50	100	**Uniform**
$\lambda = 10$						
20	0.011007	0.011534	0.011079	0.009430	0.010230	**0.010348**
40	0.002603	0.002659	0.002708	0.002498	0.002153	**0.002544**
80	0.000641	0.000643	0.000647	0.000669	0.000665	**0.000634**
120	0.000159	0.000159	0.000159	0.000161	0.000163	**0.000158**
$\lambda = 50$						
20	0.034871	0.036849	0.051606	0.166143	0.195995	**0.199678**
40	0.019341	0.009231	0.007182	0.013776	0.026343	**0.056882**
80	0.004986	0.003411	0.002974	0.001871	0.002424	**0.012398**
160	0.001423	0.001163	0.000967	0.000706	0.000551	**0.003078**
$\lambda = 100$						
20	0.071042	0.115582	0.161468	0.398213	0.440801	**0.443862**
40	0.105746	0.018387	0.02302	0.054482	0.102428	**0.194886**
80	0.020075	0.005194	0.004168	0.006272	0.008527	**0.056029**
160	0.005336	0.003147	0.002014	0.001086	0.001294	**0.012198**
$\lambda = 500$						
20	3.391497	0.631194	0.698844	0.860651	0.875699	**0.875695**
40	0.824379	0.256985	0.304312	0.481992	0.610830	**0.726789**
80	0.050871	0.061876	0.078102	0.126841	0.162180	**0.517665**
160	0.015939	0.012217	0.013647	0.020662	0.026543	**0.263567**

Table 2.6: Error of the numerical solution obtained through the direct discretization of (2.3) on Ω_p using Midpoint scheme on a non-uniform mesh generated by evaluating the solution to MMODE0 at the mesh points of a uniform mesh of N subintervals on Ω_c. The arc-length monitor function was used with various degress of monitor function smoothing for use in MMODE0. Applied to (2.3) for several λ values and several N values.

N results in a more accurate overall computation, an important property for when these methods will be applied in the setting of an error control algorithm. Ideally, a topic for future work would be the development of robust monitor function smoothing schemes, or alternatively, the development of an MMODE which includes spatial smoothing into its formulation, such as those which exist in the PDE case [34]. Note that the issue of having an effectively mapped initial computational monitor function is lessened in the time-dependent PDE case, as the transformation at a previous point in time can be used.

The arc-length monitor function is in many ways a prototype for effective monitor functions, as it captures the features that are most fundamental for an effective MT method. In the literature, many alternative monitor functions have been proposed based on factors such as curvature, interpolation error bounds, and other properties of the numerical method or the problem [4]. In order to properly use MT methods in an error control framework for general problem classes, it is important that the adaptation is driven by the use of local error estimates. The next section discusses the formulation and validation of *a posteriori* error estimation based monitor functions.

2.3 Moving Transformation Approach for BVODEs Using an Error Estimate Monitor Function

The adaptation principles established in previous sections provide us with important information on how the coupling of the solution to MMODE0 and the physical equation can be done to produce well-adapted computations. To move towards adaptive error control methods for this problem class, we require that the adaptivity be focused on regions within the domain where an error estimate associated with the computed solution of the physical BVODE is large. Therefore we wish to consider MT methods where the monitor function is based on estimates of the solution error. We want the MT method to be such that it generates a coordinate transformation that leads to stretching of the computational domain in regions corresponding to large error estimates. The basic form of an error estimate we consider is

$$E(\underline{Y}(x)) = ||\underline{Y}(x) - \bar{\underline{Y}}(x)||, \tag{2.18}$$

where $\underline{Y}(x)$ is the current approximate solution, $\bar{\underline{Y}}(x)$ is an approximate solution which has higher accuracy than $\underline{Y}(x)$, and $||\cdot||$ is a standard norm. A possible issue when working with monitor functions based on error estimates is that at some points we could have that $\underline{Y}(x) - \bar{\underline{Y}}(x) = \underline{0}$. This is problematic, as the monitor function must be strictly positive [4].

A simple error estimate-based monitor function which avoids this issue is

$$M(x) = (\epsilon_{MACH} + E(\underline{Y}(x)))^{\frac{1}{p}}, \tag{2.19}$$

where ϵ_{MACH} is the unit roundoff for the given machine and p is the order of the discretization method. The exponentiation by the reciprocal of the order of the discretization method is motivated by the equidistribution algorithm implemented in BACOLI, which makes use of similar estimates [28]. In our experimentation, we found that this kind of monitor function resulted in transformations which were more reactive to regions of high solution error. Monitor functions having approximately this form are often applied in the literature, with a constant used to ensure positivity added to a solution-dependent component which is meant to govern the adaptivity, though typically larger constants than ϵ_{MACH} are used [4]. Commonly 1 is used as the constant and an intensity parameter is used to scale the solution dependent component of the monitor function such that the coordinate transformation reacts appropriately to the solution-dependent component. We do not claim that the monitor function (2.19) is robust or is effective for use on all problems; however, for the problems considered in this thesis, it was found to be effective.

In Figure 2.11, we compare the normalized (2.19) for the problem (2.3), $\lambda = 100$, with the exact normalized error of an approximate solution $\underline{Y}(x)$, computed with the Midpoint scheme on a uniform mesh of 40 subintervals in Ω_p. From this figure, we see that the monitor function (2.19) preserves the behaviour of the regions of the highest solution error, with the exponentiation by $\frac{1}{p}$ also exaggerating regions of smaller error.

To obtain error estimates for use in the monitor function (2.19), without requiring the exact solution *a priori*, we make use of Richardson Extrapolation [13]. Richardson Extrapolation can be used to generate an error estimate by computing two solution approximations using the same discretization scheme, one using a mesh of N subintervals and one using a doubled mesh of $2N$ subintervals. (The doubled mesh is obtained by splitting each subinterval of the original mesh in half.) Note that a solution $\underline{Y}^{(N)}(x)$ generated using a pth order one-step method for BVODEs has an error that is $\mathcal{O}(h^p)$, where h is the maximum mesh subinterval length for a mesh of N subintervals. Then a solution generated using the same discretization on a mesh, $\underline{Y}^{(2N)}(x)$, has an error that is $\mathcal{O}((\frac{h_i}{2})^p) = \frac{1}{2^p}\mathcal{O}(h_i^p)$. Then note that

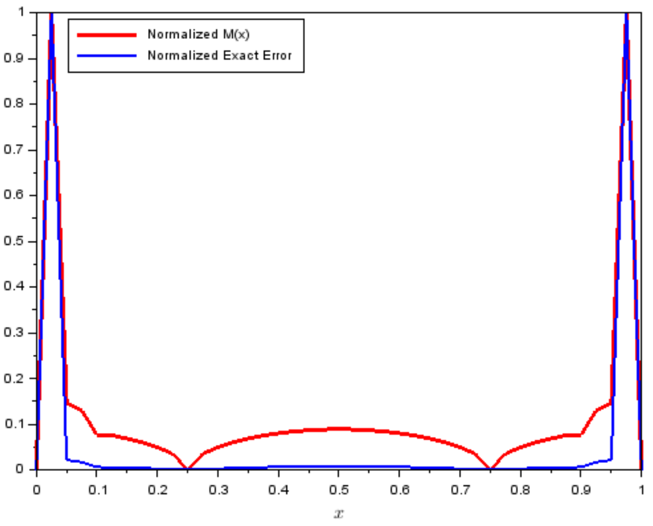


Figure 2.11: Comparison of the monitor function (2.19) with $E(Y(x))$ the exact error against the exact error, for an approximate solution to (2.3) with $\lambda = 100$, discretized with the midpoint scheme on a uniform mesh of 40 subintervals on Ω_p.

$$\underline{y}(x) - \underline{Y}^{(N)}(x) = \underline{c}h^p + \mathcal{O}(h^{p+1}), \tag{2.20}$$

$$\underline{y}(x) - \underline{Y}^{(2N)}(x) = \frac{\underline{c}}{2^p}h^p + \mathcal{O}(h^{p+1}), \quad \underline{c} \in \mathbb{R}^n, \tag{2.21}$$

where n is the number of BVODEs in the system and $\underline{y}(x)$ is the exact solution. Subtracting (2.21) from (2.20), we have that

$$\underline{Y}^{(2N)}(x) - \underline{Y}^{(N)}(x) = \underline{c}(1 - \frac{1}{2^p})h^p + \mathcal{O}(h^{p+1}), \tag{2.22}$$

and hence

$$\frac{2^p}{2^p - 1}\big(\underline{Y}^{(2N)}(x) - \underline{Y}^{(N)}(x)\big) \approx \underline{c}h^p. \tag{2.23}$$

From this we see that (2.23) gives an approximation to the leading order term in the error expression for the approximate solution $\underline{Y}^{(N)}(x)$, and therefore can serve as a viable estimate of the solution error.

Note that when using an MT method, we do not explicitly compute $\underline{Y}^{(N)}(x)$; we instead compute its analog on the computational domain, $\underline{\hat{Y}}^{(N)}(\xi)$. Let $\underline{\hat{Y}}^{(2N)}(\xi)$ be the computational analog of the solution obtained using the doubled mesh. Since the monitor function depends on the error estimate of the solution in the physical domain, we obtain the error estimate by using the evaluations of $\underline{Y}^{(N)}(x)$ and $\underline{Y}^{(2N)}(x)$ at the images of the computational mesh points mapped onto Ω_p

$$\begin{aligned} M(x(\xi_i)) &= \left(\epsilon_{MACH} + \frac{2^p}{2^p-1}||\underline{\hat{Y}}^{(2N)}(\xi_i) - \underline{\hat{Y}}^{(N)}(\xi_i)||_\infty\right)^{\frac{1}{p}} \\ &= \left(\epsilon_{MACH} + \frac{2^p}{2^p-1}||\underline{Y}^{(2N)}(x(\xi_i)) - \underline{Y}^{(N)}(x(\xi_i))||_\infty\right)^{\frac{1}{p}}. \end{aligned} \qquad (2.24)$$

These values are smoothed and then interpolated using monotonicity preserving cubic splines to provide a global *a posteriori* error estimation based monitor function which can be used to provide the beginnings of error control-based adaptation.

To demonstrate the effectiveness of this monitor in highlighting regions of large solution error, the physical BVODE was solved on a uniform mesh in Ω_p and the error estimate monitor function computed using (2.23) to obtain the error estimate. Its normalized values are plotted against the exact normalized error in Figure 2.12. Here we see this monitor function obtained using Richardson Extrapolation generates a monitor function nearly identical to the one generated using exact error values plotted in Figure 2.11.

To validate that this monitor function is effective for use in an MT method, we ran the suite of tests from the previous section, the results of which are summarized in Table 2.7. These tests were run by computing two initial low-accuracy approximate solutions, one using a uniform mesh of N subintervals and one using $2N$ subintervals. Richardson Extrapolation was then used to generate the error estimate-based monitor function and then MMODE0 was solved to obtain the coordinate

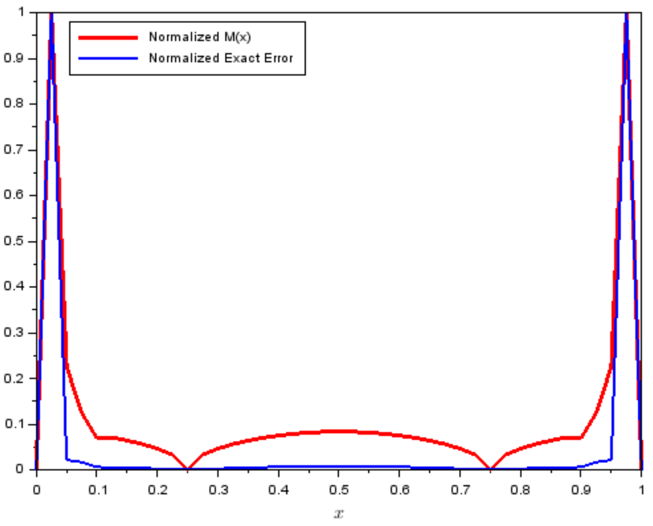


Figure 2.12: Comparison of the monitor function (2.24) with the exact error for a solution to (2.3) with $\lambda = 100$, discretized using the Midpoint scheme on a uniform mesh of 40 subintervals on Ω_p.

transformation. This transformation was then used to transform the physical BVODE such that it could be solved on a uniform mesh in Ω_c. From this data, we see patterns similar to those observed for the MT method which used the arc-length monitor function, with some degree of monitor function smoothing being required to produce well-adapted solutions. Comparing the performance of the error estimate monitor function results from Table 2.7 and the arc-length monitor function results in Table 2.5, we see that in general the arc-length monitor function and the error estimation monitor function have similar performance, with the arc-length monitor function generally performing marginally better. Even if it is true in general that the arc-length monitor produces better-adapted solutions, there is no possibility for error control when using it as the monitor function, and as such we consider this a reasonable trade-off. We also note that in general, the error estimation monitor function is sufficiently smoothed when using between 5 and 10 iterations of monitor function smoothing.

Table 2.8 gives results for the BVODE directly discretized on the mesh generated from the solution to MMODE0 obtained using the error estimation monitor function. That is, the physical BVODE (2.3) is solved on Ω_p using the non-uniform mesh obtained by using the coordinate transformation (the solution to MMODE0) to map a uniform mesh of N subintervals onto Ω_p. This is what we have been referring to as traditional r-adaptivity. Here we see similar trends as in the arc-length monitor function case, with the traditional r-refinement method performing comparably to the MT method, though the direct discretization accuracy is less sensitive to monitor function smoothing.

N	Smoothing Iterations					
	1	5	10	50	100	**Uniform**
$\lambda = 10$						
20	0.011662	0.008657	0.008713	0.010043	0.010328	**0.010348**
40	0.002652	0.002277	0.002110	0.002173	0.002331	**0.002544**
80	0.004577	0.000649	0.000757	0.000516	0.000514	**0.000634**
160	0.0015461	0.000306	0.000324	0.000145	0.000133	**0.000158**
$\lambda = 50$						
20	0.214816	0.054106	0.130663	0.191891	0.205421	**0.199677**
40	0.015989	0.010935	0.011454	0.021391	0.033984	**0.056882**
80	0.005334	0.002308	0.001921	0.002669	0.003512	**0.012398**
160	0.002232	0.000670	0.000386	0.000456	0.000522	**0.003078**
$\lambda = 100$						
20	0.216749	0.194072	0.245109	0.410908	0.442384	**0.443862**
40	0.050142	0.021612	0.032689	0.085448	0.128795	**0.194886**
80	0.010908	0.006204	0.00561	0.008834	0.012096	**0.056029**
160	0.003699	0.001354	0.001016	0.001286	0.001558	**0.012198**
$\lambda = 500$						
20	0.996382	0.887106	0.904252	0.877986	0.877337	**0.875695**
40	0.5674222	0.447365	0.467916	0.574222	0.649036	**0.726789**
80	0.959728	0.067241	0.074859	0.111337	0.143541	**0.517665**
160	0.026274	0.014657	0.014579	0.022460	0.029405	**0.263567**

Table 2.7: Error results for the MT algorithm using the error estimate monitor function (2.24), applied to (2.3) with several values of λ, N, and numbers of monitor function smoothing iterations.

The results from this section demonstrate that an error estimation-based monitor function can be applied effectively in an MT method. The benefit of using such a monitor function is the fact that it uses no heuristics to govern the adaptivity, directly adapting the solution in locations which have proven to be the most difficult for the given numerical method to solve accurately. This kind of solution adaptation is important for developing robust codes for general problems since the use of error estimation as the driver of solution adaptation is fundamental in the error control approach.

N	Smoothing Iterations					
	1	5	10	50	100	**Uniform**
$\lambda = 10$						
20	0.009265	0.008686	0.009093	0.010112	0.010327	**0.010348**
40	0.002405	0.002186	0.002115	0.002257	0.002375	**0.002544**
80	0.000845	0.000578	0.000559	0.000528	0.000538	**0.000634**
120	0.000177	0.000154	0.000145	0.000136	0.000131	**0.000158**
$\lambda = 50$						
20	0.20038	0.060465	0.141029	0.194549	0.206973	**0.199677**
40	0.009076	0.009525	0.011107	0.021498	0.034057	**0.056882**
80	0.001754	0.001826	0.001945	0.002717	0.003551	**0.012398**
160	0.000418	0.000419	0.000424	0.000481	0.000542	**0.003078**
$\lambda = 100$						
20	0.227026	0.197690	0.248302	0.411574	0.442741	**0.443862**
40	0.030310	0.020959	0.037488	0.086972	0.129675	**0.194886**
80	0.004956	0.005066	0.005452	0.008856	0.012118	**0.056029**
160	0.001061	0.001060	0.001098	0.001335	0.001590	**0.012198**
$\lambda = 500$						
20	0.9963821	0.8915974	0.9113224	0.878693	0.8777116	**0.875695**
40	0.5755018	0.4492879	0.4697822	0.5745859	0.6491942	**0.726789**
80	0.9597124	0.0685475	0.076067	0.1116563	0.1437385	**0.517665**
160	0.0160741	0.0130423	0.0147996	0.0227985	0.0296447	**0.263567**

Table 2.8: Error results for direct discretization on Ω_p using the non-uniform mesh generated through the evaluation of MMODE0 at the mesh points of a uniform mesh of N subintervals on Ω_c, where the error estimation monitor function was used in MMODE0.

2.4 Adaptive Error Control Strategy for BVODEs Using MT Methods

From the previous sections, we have demonstrated the capacity of MT methods for solution adaptivity for BVODEs. We see that in most cases, the MT methods perform significantly better than simple discretization on a uniform mesh, and perform comparably to direct discretization on approximately equidistributed mesh obtained using the coordinate transformation given by MMODE0, indicating that these methods can be an effective means for performing adaptation. To extend these approaches for use in an adaptive error control algorithm, we require an algorithm that can repeatedly refine the computation using MT adaptation until an approximate solution is generated for which an associated error estimate satisfies a given tolerance. A key aspect of this refinement is that, in addition to being able to adapt to the behaviour of the solution to the physical BVODE through the coordinate transformation, we also need to be able to change the number of subintervals employed in the mesh used as the basis for the numerical discretization process. In this section, we consider an iterative refinement procedure for BVODEs using MT methods which is used in a simple, demonstrative, adaptive error control algorithm which could be easily extended for increased efficiency and reliability.

In traditional r-refinement, and in particular in the literature on MT methods, there is little discussion on adaptively choosing the number of mesh subintervals used in the discretization. However, changing the number of subintervals in response to solution error has in the past been an essential component of adaptive error control algorithms. When N, the number of subintervals, is fixed, the best a mesh adaptation algorithm can do is to optimally position the mesh points to achieve the lowest possible error for the given N. When N is fixed, the best an MT algorithm can do is to optimally stretch the independent variable in the computational domain so that the transformed physical ODE is solved as accurately as possible for the given N using a uniform mesh on the computational domain. Therefore, we must consider methods for changing the number of mesh points in order to improve the solution accuracy.

When applying an MT method as in the previous section, we generate a coordinate transformation which transforms a physical BVODE problem onto a smoothed computational with stretching of the computational independent variable in regions where the solution error is estimated to be large. If we then suppose that the coordinate transformation $x(\xi)$ is appropriately adapted such that solution to the physical BVODE is well-smoothed in regions of predicted solution error, we can then re-use the coordinate transformation and simply solve the transformed physical BVODE for $\underline{\hat{Y}}^{(N^*)}(\xi)$, where N^* is the number of subintervals in a new uniform computational mesh. That is, once $x(\xi)$ has been used to obtain the transformed BVODE on Ω_c, it is trivial to switch from a uniform mesh in Ω_c having N subintervals to a uniform mesh in Ω_c having N^* subintervals, where N^* is chosen to reduce the error such that it is less than the tolerance. Use of this approach avoids the cost of having to update the coordinate transformation on each iteration, which incurs significant expense. Of course, there may be situations where leaving the coordinate transformation fixed may incur costs indirectly. When the coordinate transformation is re-computed using the new uniform mesh on Ω_c, the error estimates and hence the monitor function will change, and recomputing the coordinate transformation would make the adaptivity more representative of the current state of the approximate solution, meaning that the adaptivity will likely be improved and therefore a possibly significantly lower value for N can be used. By reusing the coordinate transformation, our only requirement is that the coordinate transformation is sufficiently smooth and reasonably well-adapted to the solution behaviour. This procedure is outlined in Algorithm 4, where an initial coordinate transformation and solution to the transformed equation on Ω_c are assumed. Note that this procedure greatly benefits from the use of continuous solution approximations in order to be able to have the coordinate transformation be evaluated at arbitrary points along the spatial domain.

Algorithm 4: MT Solution Update

```
1 function adaptSolution ($N^*$, $x_a$, $x_b$, $BVODE$, $x(\xi)$, $\underline{\hat{Y}}(\xi)$);
  Input  : New number of mesh subintervals $N^*$; spatial boundaries $x_a, x_b$; problem definition
           $BVODE$; pre-computed coordinate transformation $x(\xi)$; previous solution approximation
           $\underline{\hat{Y}}(\xi)$
  Output: Approximate solution on updated computation mesh $\underline{\hat{Y}}^{(N^*)}(\xi)$
2 // Generate uniform computational mesh
3 $\xi := linspace(x_a, x_b, N+1)$
4 // Solve the transformed BVODE using previous solution as an initial guess
5 $\underline{\hat{Y}}(\xi) := Y\_SOL(\xi, BVODE(x(\xi)), \underline{\hat{Y}})$
6 // Interpolate the solution.
7 $\underline{Y}(x) := interp(x(\xi), \underline{\hat{Y}}(\xi)))$
8 return $\underline{Y}(x)$
```

Algorithm 4 was applied to (2.3), with N^* chosen to be $2N$ for each new mesh to demonstrate the improvements in the approximation solutions obtained. For each test case, 5 iterations of spatial smoothing were applied, and each started from a mesh of 20 subintervals. The results from these tests are summarized in Table 2.9. Here we see that for each of the test problems the solution accuracy increases at a high rate with respect to the number of mesh subintervals. Note that when applying this method, we see that the most difficult problem, the $\lambda = 500$ case, the errors are reduced faster than is the case for the other example problems. This is likely due to the initial low-accuracy solution approximations used to generate the monitor function in this case. This means that we have a larger error estimate, which results in a coordinate transformation which is more adapted to the difficult regions of the problem, in comparison to the other problems, which have smaller error estimates.

λ	N				
	20	40	80	160	320
10	0.0086568	0.002133	0.0005313	0.0001327	0.0000332
50	0.0541063	0.0122782	0.0030012	0.0007462	0.0001863
100	0.1940718	0.029019	0.0068486	0.001689	0.0004208
500	0.8871058	0.560565	0.0928158	0.00017	0.0000426

Table 2.9: Errors after successive mesh doubling for MT method using fixed coordinate transformation generated using the error estimate monitor function.

We now propose an adaptive error control algorithm for BVODEs using MT methods, with iterative refinement achieved using Algorithm 4 in which N^* is chosen in a more adaptive manner (rather than simply choosing $N^* = 2N$). For a given error tolerance TOL, the algorithm iterates until a Richardson Extrapolation based estimate of the Global Error (GE) is beneath the tolerance. This error estimate is of the kind given in [35] and is given as

$$GE \approx \left(\frac{2^p}{2^p - 1}\right) \max_{x\in\Omega_p} \left\{ \frac{||\underline{Y}^{(N)}(x) - \underline{Y}^{(2N)}(x)||_\infty}{1 + ||\underline{Y}^{(N)}(x)||_\infty} \right\}. \tag{2.25}$$

In order to adapt the number of mesh subintervals based on the value of GE, we make use of a heuristic implemented in BACOLI [28]. As in [28], note that for our solution approximation using MT methods, we have

$$\begin{aligned} &||\hat{\underline{Y}}^{(N)}(\xi) - \hat{\underline{y}}(\xi)||_\infty = \mathcal{O}(h^p) \\ \implies &||\underline{Y}^{(N)}(x(\xi)) - \underline{y}(x(\xi))||_\infty = \mathcal{O}(N^{-p}), \end{aligned} \tag{2.26}$$

where $\underline{y}(x(\xi))$ is the exact solution and $\hat{\underline{y}}$ is its analog on Ω_c. Hence

$$GE \propto N^{-p}. \tag{2.27}$$

We require that the error estimate associated with the computed solution satisfy the error tolerance;

that is, we require

$$TOL = GE^* \propto (N^*)^{-p}. \tag{2.28}$$

Dividing (2.27) by (2.28), we have that

$$\begin{aligned} \frac{GE}{TOL} &= \left(\frac{N^*}{N}\right)^p, \\ N\left(\frac{GE}{TOL}\right)^{\frac{1}{p}} &= N^*, \end{aligned} \tag{2.29}$$

and hence the new uniform computational mesh is chosen to have N^* subintervals, where

$$N^* = N\left\lceil \left(\frac{GE}{TOL}\right)^{\frac{1}{p}} \right\rceil, \tag{2.30}$$

This subinterval selection strategy, combined with the r-adaptivity provided through the MT approach, gives us a hybrid hr-refinement algorithm which may be used to generate error-controlled solution approximations.

This algorithm was applied to our test problem using the Midpoint scheme for the spatial discretization to generate adapted solutions for modest error tolerances. Results of this can be seen in Table 2.10, where the exact error in the solution is given for varying λ, with 10 iterations of spatial smoothing used in each case. Here we see that the method is able to effectively generate solutions that are accurate to within a small multiple of the given tolerance. Additionally, the solutions require far fewer mesh subintervals to reach these errors that would be required by directly discretizing on a uniform spatial mesh. The Midpoint scheme using this method struggles to reach sharper tolerance requirements within a reasonable amount of time due to its low order of accuracy. Implementation of this algorithm using higher order methods is a topic for future work.

TOL	λ			
	10	50	100	500
Exact Error				
10^{-2}	0.0086568	0.0119317	0.0106429	0.0109345
10^{-3}	0.000986	0.001189	0.0010665	0.0013219
10^{-4}	0.000099	0.000119	0.0001221	0.0001332

Table 2.10: Exact error for solutions to (2.3) with several λ values and a simple error control algorithm which requires that the error, as estimated by (2.25), satisfy various tolerance values.

In this chapter, we have seen that MT methods have the potential for application in adaptive error control algorithms. While in practical terms, it is unlikely that such a method would be effective for BVODEs, it does provide a proof of concept for more applicable problems such as 1D and 2D PDEs. A particular point of interest is that once a coordinate transformation has been generated, this transformation can be re-used which then implies that increasing N to adapt to the error in the solution becomes the simple task of generating a courser uniform mesh in Ω_c and using the evaluations of the interpolated coordinate transformation at these points. We feel that this may be a substantial benefit when applying these methods, particularly for higher-dimensional PDE problems, an idea that has been largely under-utilized in past applications of the MT approach. In particular, this approach. combined with the additional adaptive refinement of the coordinate transformation to accelerate convergence, may lend itself to adaptive error control algorithms using MT methods and may be of particular benefit in the time-dependent PDE case.

Chapter 3

Moving Transformation Methods for PDEs

As we saw in the previous chapter, MT methods can be effectively applied within adaptive algorithms for solving BVODEs. However, we have seen that there can be substantial difficulty in effectively solving the MMODE0 due to spatial regions in which its solution exhibits rapid variation. This problem was alleviated by smoothing of the monitor function; however, this can have a negative impact on the effectiveness of the adaptivity and it is in general unclear how to generate a monitor function which *(i)* accurately describes regions of physical solution difficulty and *(ii)* is sufficiently smooth. Difficulty in solving MMODE0 has motivated the development of MT methods for the time-dependent PDE case, where these issues are alleviated: the MMPDE methods. The key observation in modern MMPDE methods is that in most cases an approximately equidistributing mesh is sufficient for use in practical computations [4]. Indeed, moving mesh software such as COLNEW and BACOLI, which both implement direct r-adaptation as a component of their adaptation algorithm, make use of de Boor's algorithm to generate fairly low accuracy approximations to an equidistributing mesh [28, 6]. In the context of time-depended PDEs, this observation is exploited through the use of time-space dependant MT equations, referred to as MMPDEs, which for a given monitor function, have a solution which converges towards an approximately equidistributing state as the MMPDE is solved forward in time [4].

MT methods for PDEs make use of a time-dependent coordinate transformation $x(\xi, t)$ so that the solution to the physical PDE can be adapted effectively for its behaviour at a given point in time. In transforming and solving a PDE using MT adaptivity there are two primary solution procedures: simultaneous and alternating procedures [4]. In the simultaneous procedure, the MMPDE and transformed physical PDE are solved together in Ω_c and the coupling between these equations is treated directly. This approach is typical for 1D problems, but the highly non-linear coupling between the transformed PDE and the coordinate transformation can result in a very stiff system of DAEs after the spatial discretization process is applied [4]. Alternating approaches decouple the solution of the coordinate transformation and the transformed PDE, solving for $x(\xi, t)$ and then $\underline{\hat{u}}(x(\xi, t), t)$ in sequence; the MT procedures for BVODEs discussed in Chapter 2 are examples of alternating approaches. Separating these two components of the computation avoids the expensive coupling but comes with its own issues, such as mesh lagging. Mesh lagging occurs when the transformation $x(\xi, t)$ is well adapted for the solution behaviour at a previous point in time but has not responded to the current behaviour of the solution [4]. It is worth noting that in the literature when an algorithm is referred to as either a simultaneous or alternating procedure, it is typically assumed to be implementing a quasi-Lagrange approach, where the coordinate transformation and solution to the physical PDE are considered to move together through time [4]. Rezoning approaches make use of alternating solution procedures; however, in this context, the coordinate transformation and the physical equation are considered to move through time independently of each other. This makes the rezoning approach closely analogous to the standard mesh adaptation approaches such as the one implemented in BACOLI [4, 5, 7]. How the distinction between the quasi-Lagrange approach and the rezoning approaches presents itself in practice will be discussed later in this chapter.

For our experiments, we consider the viscous Burgers' equation

$$u_t(x, t) = \epsilon u_{xx}(x, t) - u(x, t)u_x(x, t), \quad x \in [0, 1], \quad t > 0. \tag{3.1}$$

This is a scalar PDE which has been frequently applied in areas such as fluid dynamics. Burgers' equation is often used in validating numerical methods for PDEs due to the sharp, moving spatial layer regions which can occur in solutions to this problem. These layer regions typically require effective mesh adaptation in order for the PDE to be accurately discretized using a reasonable number of mesh points. We consider two instances of this problem, each characterized by the choice of their initial and boundary conditions:

- **OLBE**: One Layer Burgers' Equation

  This equation produces solutions having a steep wave front, with the steepness of this wavefront, and hence the difficulty of the problem, being dependent on the choice of the parameter ϵ. OLBE is defined by (3.1), coupled with the initial condition

  $$u(x,0) = 0.5 - 0.5\tanh\left(\frac{1}{4\epsilon}(x-0.25)\right), \tag{3.2}$$

  and boundary conditions

  $$u(0,t) = 0.5 + 0.5\tanh\left(\frac{1}{4\epsilon}(-0.5t-0.25)\right), \tag{3.3}$$
  $$u(1,t) = 0.5 - 0.5\tanh\left(\frac{1}{4\epsilon}(0.75-0.5t)\right). \tag{3.4}$$

  OLBE has the exact solution

  $$u(x,t) = 0.5 - 0.5\tanh\left(\frac{1}{4\epsilon}(x-0.5t-0.25)\right). \tag{3.5}$$

  An accurate approximation of this equation with $\epsilon = 10^{-3}$ obtained with BACOLI using an absolute and relative error tolerance of 10^{-6} is plotted in Figure 3.1. For our testing we consider the cases of $\epsilon = 10^{-2}$ and $\epsilon = 10^{-3}$, which are referred to as OLBEϵ2 and OLBEϵ3 respectively.

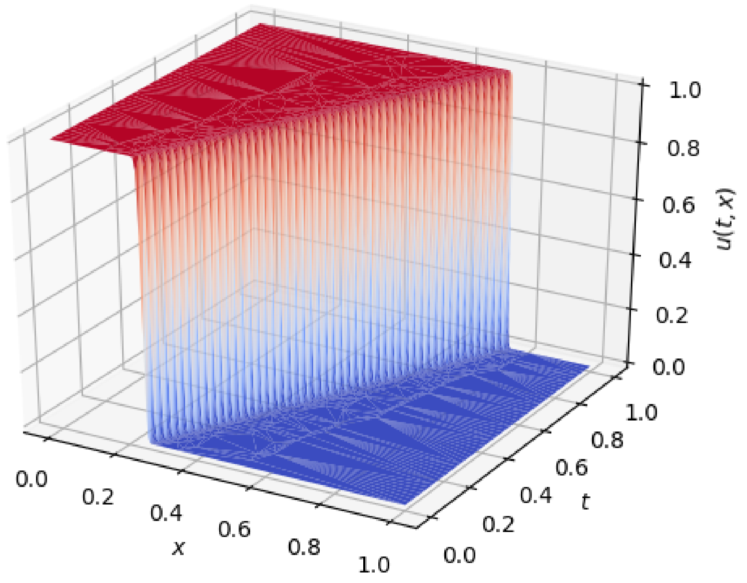


Figure 3.1: Solution to the One Layer Burgers' Equation, $\epsilon = 10^{-3}$.

- **TLBE**: Two Layer Burgers' Equation

TLBE has a solution which initially has two steep wavefronts, with the steepness of these fronts being governed by the ϵ parameter in (3.1). TLBE is defined by the initial and boundary conditions chosen such that the exact solution is given by

$$u(x,t) = \frac{0.1e^{-A} + 0.5e^{-B} + e^{-C}}{e^{-A} + e^{-B} + e^{-C}}, \tag{3.6}$$

where

$$A = \frac{0.05}{\epsilon}(x - 0.5 + 4.95t), \quad B = \frac{0.25}{\epsilon}(x - 0.5 + 0.75t), \quad C = \frac{0.5}{\epsilon}(x - 0.375). \tag{3.7}$$

This problem with $\epsilon = 10^{-3}$ was solved with BACOLI using tolerances of 10^{-6} and is plotted in Figure 3.2. For our testing we consider the cases of $\epsilon = 10^{-2}$ and $\epsilon = 10^{-3}$, which are referred to as TLBEϵ2 and TLBEϵ3 respectively.

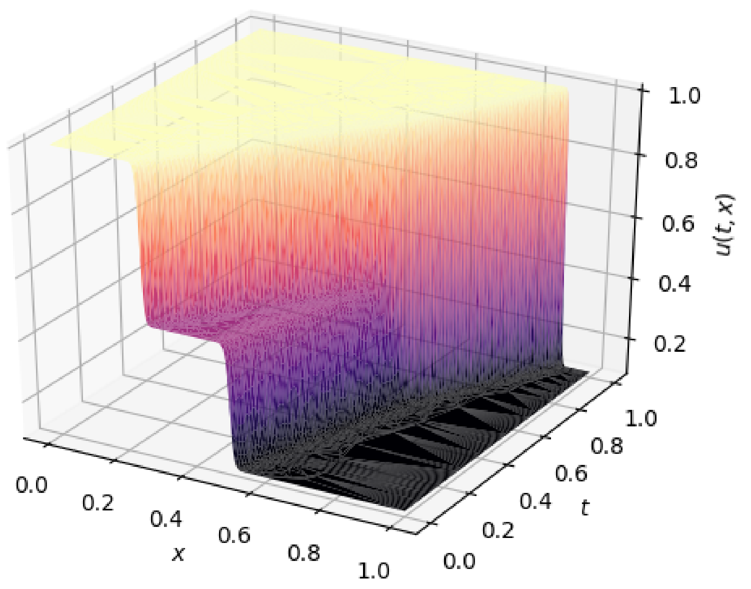


Figure 3.2: Solution to the Two Layer Burgers' Equation, $\epsilon = 10^{-3}$.

In this chapter, we consider MT methods for 1D PDEs, experimenting with and measuring various aspects of the methods and assessing the viability of these methods as adaptation algorithms for implementation in error control solvers for 1D PDEs. As in the previous chapter, the approaches considered here are those which have the potential to be extended for use in adaptive error control algorithms. This will include experimentation to determine whether MT adaptation based on the time-dependent MMPDEs can be used to effectively reduce the error in the solution to PDEs in a way which is amenable with error control. In the application of adaptive error control to time-dependent PDEs, the use of continuous solution approximations becomes invaluable; we therefore make use of continuous solution approximations in our experiments.

This chapter is organized as follows: Section 3.1 presents and discusses experiments involving the generation of time-dependent, equidistributing coordinate transformations using MMPDE5. Section

3.2 implements an MT method using a finite difference method for spatial discretization, with the arc-length monitor function governing solution adaptivity. Section 3.3 discusses error estimation based adaptation for MT methods. Section 3.4 proposes an adaptive error control algorithm for 1D PDEs which uses an alternative formulation of the MT methods and is given to demonstrate an avenue for possible future work in this area. The chapter concludes in Section 3.5 with some brief discussion of the application of MT methods to 2D time-dependent PDEs.

3.1 Computing the Coordinate Transformation

The space-time aspect of MMPDE methods is important for the application of MT methods for PDEs. MT equations for time-dependant PDEs typically implement a temporally relaxed form of the equidistribution principle. In MMPDE5,

$$x_t(\xi, t) = \frac{1}{\tau}\Big(M(x(\xi, t), t)x_\xi(\xi, t)\Big)_\xi, \quad \tau > 0, \tag{3.8}$$

the intensity parameter τ governs how quickly the coordinate transformation can react to changes in the monitor function $M(x(\xi, t), t)$, i.e., how far forward in time it must be solved until it converges to a coordinate transformation equidistributing the monitor function $M(x(\xi, t), t)$ at a given time t. While using very small values of τ seems ideal as the coordinate transformation will respond immediately to changes in $M(x(\xi, t), t)$, the choice of τ greatly affects the stiffness of the DAEs generated from the solution process applied to the MMPDE [4]. Smaller values of τ result cause the discretization of MMPDE5 to produce a DAE system that can be very difficult to solve [4, 5]. As in the case of BVODEs, some smoothing of the monitor function or some other spatial smoothing strategy is typically applied so that the MMPDE can be solved effectively [4]. Hence an MMPDE method which is both accurate and efficient must balance the quality of the adaptivity of the coordinate transformation against the spatial smoothness of the coordinate transformation in addition to the usual considerations of the accuracy of the spatial discretization and time integration methods. In the literature, there has been a large volume of work done on spatial smoothing approaches, and on effective numerical algorithms for discretization and time integration, but there has been relatively little discussion of how to choose τ or similar adaptation parameters used in other MMPDEs. The optimal choice of τ depends on the time scale of the problem, which is unknown *a priori*. In most contexts, τ has been chosen to be constant, but [4] has some discussion of adaptive refinement for the choice of τ for solving a particular class of PDE problems.

This section explores practical aspects of solving MMPDE5 in order to understand how well the resultant transformations can be computed based on several relevant performance measures. While many other MMPDEs for 1D MT methods have been derived and applied, we focus our attention on MMPDE5 (3.8) since it is a popular choice for MT methods. MMPDE5 is a parabolic PDE which means that we can apply standard error control software to solve it, and the approach to adaptivity which is obtained through the use of MMPDE5, with its temporal relaxation parameter, is one which has been effectively applied to higher dimensional MT methods [5]. For this task we make use of BACOLI so that MMPDE5 can be solved to within prescribed accuracy requirements, allowing us to also understand the role that the accuracy of the solution to MMPDE5 plays in generating coordinate transformations that correspond to good quality equidistribution.

For our experiments in this section, our choice of monitor function is the exact arc-length monitor function

$$M(x(\xi,t),t) = \sqrt{1 + u_x(x(\xi,t),t)^2}, \tag{3.9}$$

where the evaluations of $u_x(x(\xi,t),t)$ are obtained by using the exact solutions to OLBE and TLBE. This monitor function is a popular choice in MT methods for PDEs, since as in the BVODE case, regions in the spatial domain where the solution to the differential equation varies rapidly are difficult to discretize accurately, potentially leading to large error contributions from the spatial discretization component of the Method of Lines algorithm. Therefore, adapting the computation by smoothing the physical PDE in regions where the arc-length is large will smooth the solution of the transformed physical PDE in the corresponding regions of Ω_c allowing for a more accurate discretization on the uniform mesh employed in Ω_c.

In order to evaluate the quality and efficiency of computed solutions to MMPDE5, we consider two performance metrics. The first is E_{eq}, introduced in Chapter 2, which measures how well $x(\xi, t)$ succeeds in generating an equidistributed mesh of N subintervals in Ω_p. E_{eq} is a measurement of the departure of a coordinate transformation, $x(\xi, t)$, from equidistribution at some point in time and is obtained by measuring how well equidistributed the mesh of N subintervals it generates is. E_{eq} is given in the PDE case by

$$E_{eq}(x(\xi,t),t) = \max_{i=1,\dots,N} \left\{ \left| \frac{\int_{x(\xi_i,t)}^{x(\xi_{i+1},t)} M(x,t)dx - \frac{\sigma}{N}}{\frac{\sigma}{N}} \right| \right\} \tag{3.10}$$

with

$$\sigma = \int_{x_a}^{x_b} M(x,t)dx. \tag{3.11}$$

The second performance metric is CPU time, which in each case is reported as the median of five runs.

As in Section 2.1, we wish to understand the practical algorithmic aspects of solving MMPDE5 to better understand how to effectively generate equidistributing coordinate transformations obtained as the solution to this MMPDE. Particular points of interest include the effects of temporal relaxation on the solution to MMPDE5, as controlled by, τ and the effects of the monitor function smoothing done through iterations of the discrete smoothing scheme (2.6) discussed in Chapter 2. These effects are evaluated in terms of their impact on the quality of the resultant coordinate transformation as measured by E_{eq} and on the efficiency of the computation in terms of CPU time. Another point of interest is how accurate the overall computation must be in order to produce a coordinate transformation of reasonable quality, i.e., how does the global error in the solution to MMPDE5 affect the quality of the equidistribution. While we of course understand that a loss of monotonicity can result in catastrophic error (and thus it is essential that a computed solution to an MMPDE be sufficiently accurate to preserve monotonicity), in an efficient MT method it is desirable to be able to compute a coordinate transformation which is reasonably accurate for minimal cost so that the cost of the overall computation is not dominated by the cost of the adaptation process.

To perform these experiments, we applied BACOLI [7] to solve MMPDE5 for the exact arc-length monitor functions induced by each of our test problems. The driver program and necessary utility functions were implemented in Fortran 95. Note that BACOLI solves MMPDE5 using a non-uniform adapted mesh generated through de Boor's algorithm, which is in contrast with typical approaches used to discretize MMPDE5, where a uniform mesh on the computational domain is employed. (At this point in our investigation, where we are studying the role of the solution of MMPDE5, this is not an important consideration.) The monitor function provided to MMPDE5 is a monotonicity preserving cubic spline interpolating discrete, smoothed evaluations of the exact arc-length monitor function at 160 points uniformly distributed on Ω_p. This number of evaluations was chosen based on experimental results which indicated that it was a good choice for minimizing the effect of interpolation error. (In a more general setting, the number of evaluations of the monitor function would have to be done using a heuristic which could adapt to the difficulty of the PDE.) As in the BVODE case, the smoothed analog of this monitor function on Ω_c, generated after each time step, is used. This smoothed monitor function is defined as $\hat{M}(\xi, t) = M(x(\xi, t), t)$, where $x(\xi, t)$ is the coordinate transformation which is available after each time step, when the monitor function is updated. Therefore this transformed monitor function is defined in terms of the previous coordinate transformation, which changes the formulation of MMPDE5 to depend on this previous coordination, a deviation from the previously stated mathematical theory for these methods. Note that in most practical contexts the coordinate transformation does not change greatly between two consecutive time steps, and hence the results obtained through the use of a monitor function transformed in this way will give results which are very similar to those seen for the standard formulations of these methods.

Since the monitor function is updated and fixed before each time step when solving MMPDE5 in a MOL procedure, in order for the $x(\xi, t)$ to be close to equidistributing at each point in time, it may have to react quickly to changes in the monitor function. In typical MT methods, we are solving the MMPDE and physical PDE in either a simultaneous or alternating fashion and are guiding the coordinate transformation through time towards equidistribution, with τ determining how far the equation must be solved in time to converge to an equidistributed state. This means that $x(\xi, t)$ may well be lagging behind the optimal $x(\xi, t)$ by several time steps, catching up only if the solution to the physical PDE stops changing with time, or never catching up if the solution to the physical PDEs continues to change with time, such as for OLBE and TLBE.

For each of the test problems, MMPDE5 was solved with several choices for the parameters of the time-dependant MT problem. The results of this testing in terms of the E_{eq} values attained in these test cases can be seen in Table 3.1, which were generated by solving the MMPDE5 to $t_{out} = 1$ using BACOLI with absolute and relative error tolerances of 10^{-6}. To compute E_{eq}, evaluations of the solutions to MMPDE5 computed by BACOLI were evaluated at the mesh points of a uniform mesh on Ω_c having 160 subintervals. Table 3.1 also includes the E_{eq} values for obtained for meshes of 160 subintervals generated using de Boor's algorithm and for a uniform mesh based on the monitor function (3.9). For the spatial error estimation and control, we set BACOLI to use its local extrapolation spatial error estimation scheme and set the number of collocation points per subinterval, *kcol*, to 4, corresponding to the use of a continuous approximation in terms of fifth order B-spline basis polynomials. From this data, we can make several observations.

- For the problems using the smaller choice for the parameter ϵ, which corresponds to the presence of sharp spatial layer regions in the solution to these PDEs, the coordinate transformation is more poorly adapted. This is likely due to the lag in the coordinate transformation as it cannot adapt quickly enough to the rapid changes in the solution which occurs in these problems. While this is not a factor in direct r-refinement algorithms for 1D problems as a mesh which equidistributes a given monitor function can always immediately be generated, it must be taken into account when implementing a time-dependent MT method.

	Smoothing Iterations					
	1	5	10	50	de Boor	Uniform
$\tau = 10^{-2}$						
OLBEϵ2	0.0386	0.1168	0.1724	0.3472	0.0686	5.781
OLBEϵ3	0.4593	0.6570	0.7214	0.8389	0.8383	36.12
TLBEϵ2	0.0664	0.1559	0.2227	0.4179	0.0546	4.635
TLBEϵ3	0.4292	0.6211	0.6945	0.8196	0.8048	41.35
$\tau = 10^{-1}$						
OLBEϵ2	0.0413	0.1223	0.1781	0.3479	0.0686	5.781
OLBEϵ3	0.4247	0.6532	0.7246	0.8395	0.8383	36.12
TLBEϵ2	0.0595	0.1491	0.2179	0.4156	0.0546	4.635
TLBEϵ3	0.4243	0.6219	0.6960	0.8204	0.8048	41.35
$\tau = 10^{-0}$						
OLBEϵ2	0.0889	0.1509	0.2024	0.3648	0.0686	5.781
OLBEϵ3	0.4541	0.6543	0.7259	0.8405	0.8383	36.12
TLBEϵ2	0.0991	0.1089	0.1651	0.3882	0.0546	4.635
TLBEϵ3	0.4015	0.6102	0.6919	0.8219	0.8048	41.35

Table 3.1: E_{eq} for varying problems and parameter choices for solving MMPDE5 with BACOLI. E_{eq} was computed using $N = 160$.

- τ determines how well-adapted the coordinate transformation is, with lower τ values corresponding to lower E_{eq} values. This is expected as for lower values of τ the solution to MMPDE converges more rapidly to equidistribution, meaning that the coordinate transformation does not lag as far behind the behaviour of the monitor function, and thus the solution to the physical PDE.

- We see that in all cases, spatial smoothing moves the coordinate transformation further from equidistributing, with this effect becoming increasingly pronounced as more iterations are done. This is expected since the smoothing of the monitor function changes its behaviour, moving it further away from that of the original monitor function.

- The E_{eq} value for the MT test cases considerably outperforms the value for a uniform spatial mesh, and if few iterations of spatial smoothing are applied and τ is chosen to be small, the MMPDE generated coordinate transformation can perform better than the mesh generated using de Boor's algorithm. This is possible since de Boor's algorithm generates a fairly low-accuracy approximation to the coordinate transformation, and the MMPDE is solving a fully continuous form of the equidistribution principal, potentially having a higher-accuracy approximation to the monitor function.

To understand how the choice of MT method parameters affects the efficiency in solving MMPDE5, in Table 3.2 we provide timing results for each of these test cases considered in Table 3.1. Here we see that, as expected, the choices of τ and the number of iterations of spatial smoothing greatly affects the efficiency of solving MMPDE5. As in the case of solving MMODE0, in our experimentation, we noted very little increase in the quality of the coordinate transformation when MMPDE5 was solved to particularly stringent tolerances (this is not shown here but was observed over the course of our experimentation), though we did find that a higher degree of monitor function smoothing was necessary in the low tolerance cases in order to preserve monotonicity. This agrees with the approach often taken in the literature of using a lower order discretization to solve the MMPDE and a higher order method for discretizing the physical PDE, which is reasonable as the objective of the method is to accurately solve the target PDE, as opposed to solving the MMPDE accurately. Note that some smoothing seems to be required to get good performance in terms of CPU time, but when many smoothing iterations are used, the performance worsens due to the cost of iterating the monitor function smoothing algorithm many times on each time step.

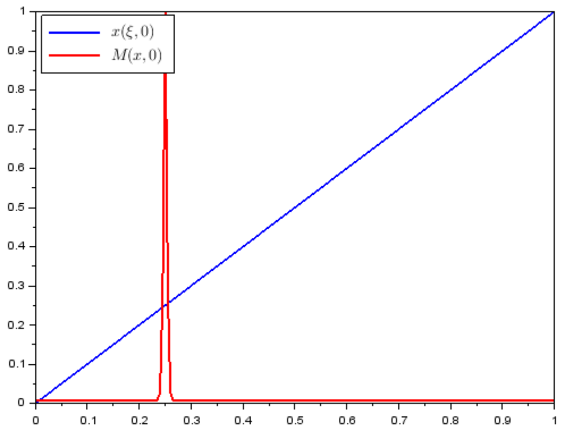
	Smoothing Iterations			
	1	5	10	50
$\tau = 10^{-2}$				
OLBEϵ2	2.641	2.703	2.844	4.906
OLBEϵ3	21.53	15.31	17.77	28.17
TLBEϵ2	2.906	3.141	3.516	5.172
TLBEϵ3	15.73	12.73	14.89	27.50
$\tau = 10^{-1}$				
OLBEϵ2	2.297	2.250	2.172	3.047
OLBEϵ3	6.563	4.125	3.938	7.891
TLBEϵ2	1.813	1.563	2.094	2.734
TLBEϵ3	6.797	4.703	4.031	6.375
$\tau = 10^{-0}$				
OLBEϵ2	1.422	1.203	1.125	1.406
OLBEϵ3	3.703	2.703	2.500	2.422
TLBEϵ2	0.859	0.844	0.844	1.250
TLBEϵ3	3.938	2.859	2.859	2.891

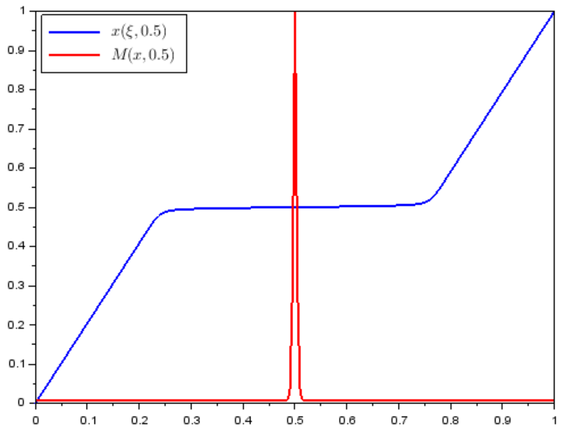
Table 3.2: CPU time for varying parameter choices when solving MMPDE5 with BACOLI.

For OLBE with $\epsilon = 10^{-3}$, Figure 3.3 plots the coordinate transformation monitor function at several points in time for a computation using 5 iterations of monitor function smoothing and $\tau = 10^{-2}$. In all but the first time instance $t = 0$, the coordinate transformation can be seen to provide reasonable adaptation, providing a mapping which corresponds to effectively clustering uniformly spaced points on Ω_c into regions in Ω_p where $M(x(\xi, t), t)$ is large. At $t = 0$, the lack of adaptivity is a consequence of the initial condition for MMPDE5 being the simple uniform transformation $x(\xi, t) = \xi$. While we see that the transformation eventually becomes well-adapted, if the equation initially has features which are difficult to discretize on a uniform mesh, this initial lack of adaptivity can become a source of either error or inefficiency. Without adaptation through the coordinate transformation, a large number of mesh points will likely be required in order to accurately discretize the problem until it has been solved far enough forward in time for $x(\xi, t)$ to be well adapted to the problem. If not enough points are used at this initial time this could contribute a large amount of spatial error which could then be propagated through the solution when integrating the PDE through time, leading to globally catastrophic results. In each of our test problems, the sharp layer regions present at $t = t_0$ make this a particularly relevant facet of the computation.

This issue can be resolved by initially solving the MMPDE through a "virtual time" interval $[t_0, t_0 + \delta], \delta > t_0$, using a fixed initial monitor function based on the solution behaviour at t_0, i.e., the initial conditions. This virtual time integration will result in a coordinate transformation which is well-adapted to the initial behaviour and which can be used as an effective initial condition for the MMPDE by taking $x_0(\xi) = x(\xi, t_0 + \delta)$. When using heuristic monitor functions such as those based on arc-length, curvature or *a priori* error bounds for the numerical method, the initially fixed monitor function can be obtained from the initial condition of the physical PDE, $u_0(x)$ [4]. In the case of a monitor function derived from *a posteriori* information, and in particular error estimation based monitor functions, a possibility is to derive the initial monitor function by solving the untransformed physical PDE on Ω_p forward for a small time interval and then use the estimated error in this initial approximation to generate the monitor function. The authors of MOVCOL suggest the use of such an approach when solving problems with difficult initial behaviour [32]. *This idea can also be extended in certain MT algorithms to be applied on a per-step basis; for each monitor function we* *integrate the MMPDE through virtual time to generate a well-adapted coordinate transformation on each time step, allowing a larger value for τ to be used and also avoiding the implicit assumption that the monitor function does not change too rapidly on any particular time interval.*

The experimentation in this section has provided valuable information which we will be able to apply in the proceeding sections to perform adaptation on PDEs using the MT adaptation approach.





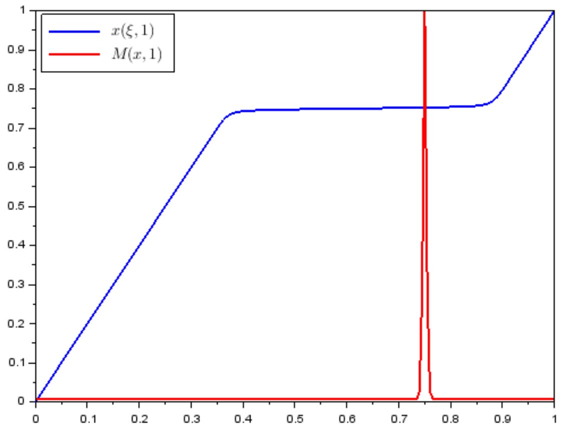


Figure 3.3: Coordinate transformation and normalized arclength monitor function at several points in time for OLBEϵ3, $t = 0$ (top), $t = \frac{1}{2}$ (middle), $t = 1$ (bottom). 5 iterations of monitor function smoothing, $\tau = 10^{-2}$. The spike in $M(x,t)$ corresponds to the location of the travelling layer region in the solution to OLBEϵ3.

3.2 Moving Transformation Approach for PDEs Using an Arc-Length Monitor Function

As in the previous chapter, we begin our exploration in applying MT methods to PDEs by first implementing adaptation based on the arc-length monitor function. Whereas in the context of BVODEs the arc-length monitor function was generated using some previously obtained solution approximation, in the context of PDEs, we derive this monitor function by using the arclength of the solution at the previous time step, a fairly standard approach in r-refinement approaches [4]. To discretize the spatial domain of MMPDE5 and the transformed physical PDE, we use an implementation of the second-order finite difference discretization methods (1.7)-(1.9) described in Chapter 1. Note that this finite difference scheme assumes the use of a uniform mesh. In the context of MT methods, the use of numerical methods which assume a uniform mesh is always acceptable, as the main idea is that the differential equation will be smoothed such that it can be solved effectively on the uniform mesh in Ω_c. Generally, an advantage of MT methods is that simple discretization schemes which assume a uniform mesh can be applied, as opposed to the requirement of using non-uniform discretization schemes in standard r-refinement methods. This is of particular utility when solving higher dimensional problems, as the use of simple rectangular grids on Ω_c allows one to avoid the use of complicated stencils and associated data structures which are commonly used when solving these problems on non-uniform grids in higher-dimensions.

The implementations discussed in this and all the subsequent sections make use of the method of lines solution procedure and are implemented in MATLAB. To solve the systems of DAEs resulting from the spatial discretization, we use the *ode15i* solver [25], which computes error-controlled solutions to general fully implicit DAE systems. The temporal error contributed when solving the DAEs is controlled using the *ode15i* ATOL and RTOL parameters, which control the absolute and relative solution errors respectively. These tolerances are set to be relatively sharp in these computations to ensure that the most significant source of error in these computations will be the error contributed from the spatial discretization. Note that, as in the MT experiments in the previous chapter, we interpolate the solution to the transformed physical PDE using Hermite cubic splines and the solution to MMPDE5 using monotonicity preserving cubic splines. In each case this produces continuous solution approximations of accuracy $\mathcal{O}(h^2)$, since the orders of accuracy of the Hermite and monotonicity preserving cubic splines are, respectively, $\mathcal{O}(h^4)$ and $\mathcal{O}(h^3)$, which means that the interpolation errors are dominated by the errors in the discrete solution approximation.

As mentioned previously, when using an MT method we have the choice of three main approaches, each corresponding to different methods of solving the non-linear, coupled system consisting of the transformed physical PDE on Ω_c and the MMPDE on Ω_c, which have the following forms,

$$\begin{aligned}\hat{u}_t(\xi,t) =& f\left(x(\xi,t),t,\hat{u}(\xi,t),\frac{\hat{u}_\xi(\xi,t)}{x_\xi(\xi,t)},\frac{1}{x_\xi(\xi,t)}\Big(\frac{\hat{u}_\xi(\xi,t)}{x_\xi(\xi,t)}\Big)_\xi\right)+\frac{\hat{u}_\xi(\xi,t)}{x_\xi(\xi,t)}x_t(\xi,t),\\ x_t(\xi,t) =& \frac{1}{\tau}\Big(M(x(\xi,t),t)x_\xi(\xi,t)\Big)_\xi. \end{aligned}\tag{3.12}$$

The simultaneous solution procedure is characterized by solving both PDEs in the above system together using the quasi-Lagrange approach, handing the coupling between the components of this system directly [4]. The alternating procedure also applies the quasi-Lagrange approach; however, the MMPDE and the transformed physical PDE are solved in sequence, first updating the coordinate transformation by integrating the MMPDE for a time-step and using the resultant coordinate transformation to then transform the physical PDE to Ω_c and further advance its solution through time. The rezoning approach works in an alternative way, one that is similar to the mesh refinement algorithms implemented in existing 1D PDE solvers. It considers $x(\xi,t)$ to be fixed in time whenever the physical PDE is to be discretized, which is expressed mathematically by setting $x_t = 0$ in (3.12). Discretization using this approach is typically done directly on Ω_p, allowing an approximation of the physical solution $u(x(\xi,t),t)$ to be obtained directly; however discretizations on Ω_c have been implemented, such as a finite difference method for 2D problems demonstrated in [4]. A consequence of this decoupling of the movement of the coordinate transformation and solution to the physical PDE is that interpolation of solutions between the different coordinate transformations is required each time the coordinate transformation is integrated in time.

In the previous section, we discussed how evolving $x(\xi,t)$ from the initial uniform transformation will cause $x(\xi,t)$ to be poorly adapted at the beginning of the computation, potentially leading to either inefficiency or error. Recall also the notion of integrating the MMPDE through virtual time, where the MMPDE is solved forward in time for the sole purpose of ensuring that the resultant coordinate transformation at a given point in physical time is well-adapted to the monitor function; this virtual time integration has no connection to the physical time of the PDE being solved. To avoid the phenomenon of the coordinate transformation being initially poorly adapted to the solution, an fixed initial monitor function $M(x,t_0)$ (generally derived either using the known initial conditions of the PDE problem, $u_0(x)$, or by solving the PDE through some small time interval and using information from this solution such as an error estimate) is used to solve the MMPDE through a virtual time interval, $[t_0,t_v]$, which is sufficiently large for it to converge to an approximately equidistributing coordinate transformation. The transformation $x(\xi,t_0+t_v)$ is determined to have converged when the condition

$$\max_{\xi\in\Omega_c}\{|x(\xi,t_0+t_v)-x(\xi,t_0+(t_v-\delta))|\} < \min\{ATOL, RTOL\}, \tag{3.13}$$

where t_v is the current point in virtual time and δ is the size of the previous time step. Explained more intuitively, (3.13) imposes the condition that the coordinate transformation will be sufficiently close to equidistribution when the difference between the transformation available at any two consecutive points in time is negligible; this is motivated by similar criteria used in [4]. This process results in an initially well-adapted initial condition to the MMPDE, $x_0^{(v)}(\xi)=x(\xi,t_0+t_v)$. We can then generate effective initial conditions for the system (3.12) of the form

$$
\begin{aligned}
x(\xi, t_0) = x_0(\xi) &:= x_0^{(v)}(\xi), \\
\hat{u}(\xi, t_0) &= u_0(x_0(\xi)),
\end{aligned}
\tag{3.14}
$$

where u_0 is the function providing the initial conditions for the PDE to be solved. This allows initial adaptivity for problems with difficult solution behaviour near t_0.

Solution procedures implementing MT methods for PDEs can be constructed in a wide variety of ways depending on the application and intended use of the method. An example of an MT algorithm is Algorithm 5, which describes an approach to MT adaptation for PDEs which assumes the use of a fixed uniform mesh on Ω_c of N subintervals, $\{\xi_i\}_{i=1}^{N+1}$. This algorithm uses a continuous monitor function which is held fixed at the beginning of each time step; this monitor function uses some initial solution information that is assumed to be available. Note that in line 12 of Algorithm 5, the processes of updating the coordinate transformation and the transformed PDE are combined into one step; this is to demonstrate that these can be done in either an alternating or simultaneous fashion.

For our computational experiments in this section, we limit ourselves to using the simultaneous solution procedure, which is commonly implemented for 1D problems and largely avoids issues such as lag in the coordinate transformation at the expense of requiring the highly non-linear coupling of the MMPDE and the transformed PDE to be handled directly. This coupling is a potential inefficiency due to the stiffness of the DAEs resulting from spatial discretization procedure [4] (the time integration for a stiff DAE system is of higher computational cost than that of a non-stiff system). We generate the well-adapted initial conditions (3.14) using a fixed arc-length monitor function derived from the initial conditions and then we solve the MMPDE through virtual time, starting from the uniform transformation $x(\xi, t) = \xi$, until the convergence criterion (3.13) is met. After these initial conditions are obtained and we begin the task of solving the coupled system (3.12), the monitor function is updated after each time step. Note that a smoothed, computational

Algorithm 5: Basic MT PDE algorithm.

```
1  function basicMTPDE ($N$, $x_a$, $x_b$, $t_0$, $t_{out}$, $MMPDE$, $PDE$, $uinit$, $U_0$, $\xi$, $ATOL$, $RTOL$);
   Input  : Number of mesh subintervals $N$; spatial boundary points $x_a, x_b$; starting time $t_0$; output
            time $t_{out}$; problem definition $PDE$; the initial conditions for $PDE$, $uinit$; initial solution
            information for monitor function construction $U_0$; uniform computation mesh $\xi$; absolute
            and relative error tolerances in time integration $ATOL$ and $RTOL$
   Output: Approximate solution $U(x, t_{out})$
2  // Initial uniform transformation.
3  $x(\xi, t_0) := \xi$
4  // Generate the initial continuous monitor function based on solution information, defined on $\Omega_c$.
5  $\hat{M}(\xi, t) := MonitorGen(x(\xi, t_0), U_0(x, t_0)$
6  // Integrate the transformation through virtual time until convergence condition (3.14) is met
7  while not $TransformationConverged(x(\xi, t_0), ATOL, RTOL)$ do
8      $x(\xi, t_0) := SolveX(x(\xi, t_0), \hat{M}(\xi, t))$
9  end
10 // Generate appropriate IC's for $\hat{U}$ using adapted $x(\xi, t_0)$
11 $\hat{U}_0 := uinit(x(\xi, t_0))$
12 // Solve the coupled system to $t_{out}$
13 $t := t_0$
14 while $t < t_{out}$ do
15     $\begin{cases} x(\xi, t+\delta) := SolveX(\xi, x(\xi, t), t, M(x, t)) \\ \hat{U}(\xi, t+\delta) := Solve\hat{U}(\xi, \hat{U}, x(\xi, t+\delta), t) \end{cases}$
16     $\hat{M}(\xi, t+\delta) := MonitorGen(x(\xi, t), \hat{U}(\xi, t+\delta))$
17     $t := t + \delta$
18 end
19 // Map the computed solution on $\Omega_c$ back to $\Omega_p$
20 $U(x, t_{out}) := ChangeDomain(x, \hat{U})$
21 return $U(x, t_{out})$
```

arc-length monitor $\bar{M}(\xi, t)$ is used in practice, which corresponds to $M(x(\xi, t), t)$. As we saw in the previous section, spatial smoothing can help to make the coordinate transformation solvable in an efficient manner. In the literature, it is often recommended that approximately 5 iterations of the spatial smoothing scheme are used when generating the monitor function, and we expect that due to the temporal relaxation of the equidistribution principle implemented in MMPDE5 that a lower degree of smoothing will be required in general [4, 5].

To succinctly demonstrate the effectiveness of Algorithm 5 for solving a PDE using the simple second order finite difference discretization (1.7)-(1.9) on a uniform computational mesh, Figure 3.4 shows a computed solution to OLBEϵ3 with $N = 20$, the MMPDE τ parameter set to 10^{-2}, and 5 iterations of monitor function smoothing used. Note that for this and all further tests, *ode15i* is used with absolute and relative error tolerances of 10^{-5}. Figure 3.5 shows a solution obtained using a direct discretization on a uniform mesh in Ω_p with no spatial adaptivity, and Figure 3.6 gives the exact solution. Here we see that using the MT method applied with this finite difference method substantially improves the ability of the method to accurately handle the steep layer regions present in the solution to OLBE, particularly in the $\epsilon = 10^{-3}$ case, where the propagation of discretization errors typically has a catastrophic effect on the solution error. Error results for the case where $\tau = 10^{-2}$ are given in Table 3.3 and for the $\tau = 10^{-1}$ case in Table 3.4. The errors here are $\max_{i=1,\ldots,N+1}\{|U(x(\xi_i,t),t) - u(x(\xi_i,t),t)|\}$, where $\{\xi_i\}_{i=1}^{N+1}$ is a uniform mesh on Ω_c of N subintervals, $U(x(\xi,t),t)$ is the approximate solution, and $u(x(\xi,t),t)$ is the exact solution. In these tables, we give results for each of the test PDEs considered at the beginning of the chapter: OLBEϵ2, OLBEϵ3, TLBEϵ2 and TLBEϵ3. In each of these test cases, we vary the number of mesh points used for spatial discretization and the number of iterations of the monitor function smoothing iterations. Results from the uniform mesh case are also considered for reference.

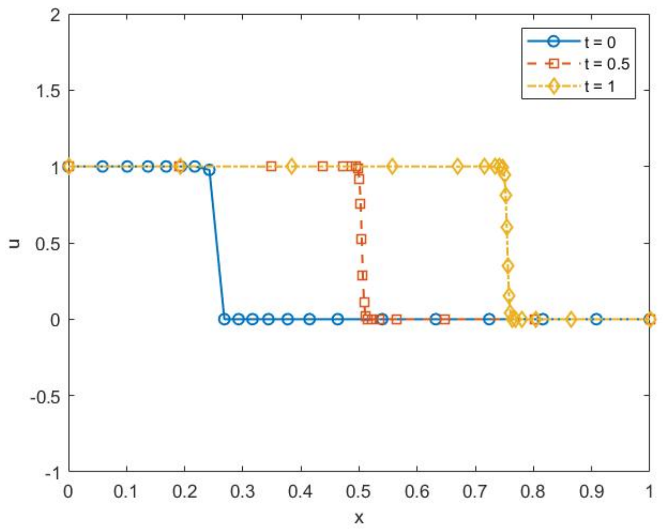


Figure 3.4: OLBEϵ3. Solved using finite difference discretization (1.13) using a MT method with arc-length monitor function, $\tau = 10^{-2}$, $N = 20$ subintervals, and 5 iterations of monitor function smoothing.

Interpreting the data from these tables, we first see that the solutions obtained using the MT method with the second order finite difference schemes for spatial discretization consistently outperform those obtained by simply solving the PDE on a uniform mesh of the same number of subintervals in Ω_p. For example, in the OLBEϵ3 and TLBEϵ3 cases, the error results obtained using the MT algorithm with a computational mesh of only $N = 20$ mesh points outperforms even the $N = 80$ case for the cases of direct discretization on a uniform mesh. In general, we see that sufficient smoothing of $M(x,t)$ is required in order to generate the most accurate solutions possible. However, as we saw in Chapter 2 in the context of MT methods for BVODEs, smoothing the monitor

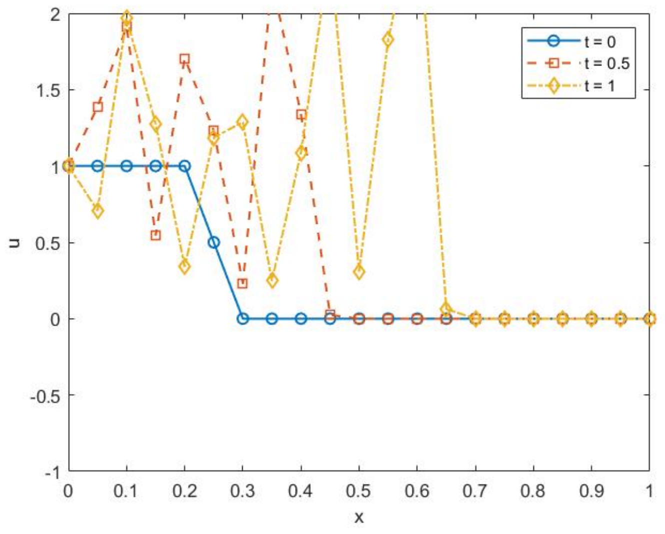


Figure 3.5: OLBEϵ3. Solved directly using finite difference discretization (1.13) on a uniform mesh with $N = 20$ subintervals.

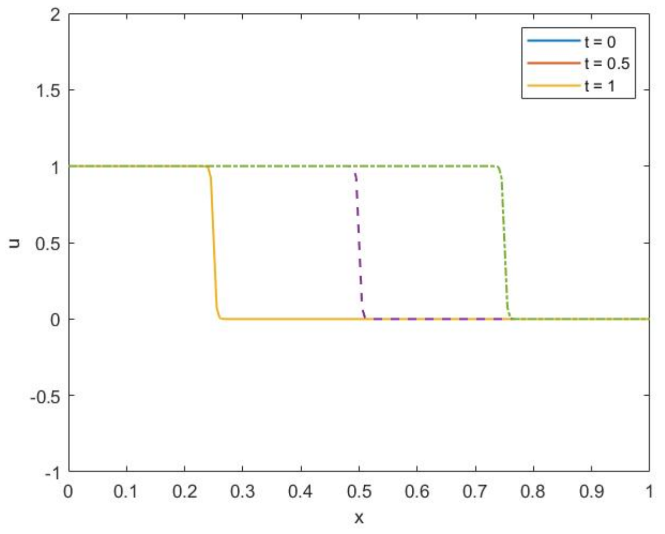


Figure 3.6: OLBEϵ3. Exact Solution

function excessively results in higher solution error, with the error approaching that of the direct uniform discretization case. This is due to the fact that the monitor function develops near-constant behaviour, and hence the resultant coordinate transformation is close to the uniform transformation $x(\xi, t) = \xi$. Further, if the number of mesh points used to discretize the coupled system it too small, and many iterations of monitor function smoothing are applied, we see that for certain problems fatal errors in the computation occur, which correspond to *ode15i* failing to solve the DAE system that arises from the discretization process. These fatal errors are due to rapidly growing spatial

	Smoothing Iterations			
	5	10	50	Uniform
$N = 20$				
OLBEϵ2	0.0148	0.0227	0.1943	0.2452
OLBEϵ3	0.5447	0.8905	ERR	1.715
TLBEϵ2	0.0163	0.0196	0.0694	0.1599
TLBEϵ3	0.0509	0.3300	ERR	1.3336
$N = 40$				
OLBEϵ2	0.0060	0.0046	0.0068	0.0535
OLBEϵ3	0.0315	0.0076	0.2959	1.9738
TLBEϵ2	0.0059	0.0047	0.0052	0.0370
TLBEϵ3	0.0089	0.0087	0.2611	0.9821
$N = 80$				
OLBEϵ2	0.0025	0.002	0.0009	0.0126
OLBEϵ3	0.0216	0.0115	0.0025	1.1621
TLBEϵ2	0.0020	0.0017	0.0009	0.0096
TLBEϵ3	0.007	0.0041	0.0016	0.7734

Table 3.3: Error results for arc-length monitor function MT method, $\tau = 10^{-2}$.

	Smoothing Iterations			
	5	10	50	Uniform
$N = 20$				
OLBEϵ2	0.0162	0.0236	0.1907	0.2452
OLBEϵ3	0.4445	0.6429	ERR	1.715
TLBEϵ2	0.0172	0.0201	0.0714	0.1599
TLBEϵ3	0.1531	0.3115	ERR	1.3336
$N = 40$				
OLBEϵ2	0.0073	0.0052	0.0068	0.0535
OLBEϵ3	0.21	0.0763	0.6275	1.9738
TLBEϵ2	0.0061	0.0048	0.0057	0.0370
TLBEϵ3	0.0561	0.0273	0.2483	0.9821
$N = 80$				
OLBEϵ2	0.0033	0.0024	0.0011	0.0126
OLBEϵ3	0.1651	0.0905	0.0159	1.1621
TLBEϵ2	0.0021	0.0017	0.00084	0.0096
TLBEϵ3	0.0503	0.0266	0.0042	0.7734

Table 3.4: Error results for arc-length monitor function MT method, $\tau = 10^{-1}$.

error contaminating the solution due to poor adaptivity, causing catastrophic effects in the solution error in the same way as was seen in Figure 3.5, causing *ode15i* to be unable to meet its tolerance requirements.

Comparing the error results in Tables 3.3 and 3.4, we observe that, as expected, the computations where τ is chosen to be small typically exhibit lower error. This is expected since the MMPDE is providing a higher degree of adaptivity as it responds to changes in the monitor function nearly immediately, whereas in the $\tau = 10^{-1}$ case the coordinate transformation consistently lags behind the behaviour of the monitor function thus providing inferior adaptivity. For the more difficult problems, OLBEϵ3 and TLBEϵ3, this effect can be very pronounced. As we saw previously, smaller choices of τ do come at the price of the coordinate transformation being substantially more expensive to compute accurately. Therefore in practice, it may be more effective in these simultaneous procedures to choose τ to be somewhat large, but make use of more mesh points when discretizing the transformed physical PDE. Due to the effectiveness of the choice of $\tau = 10^{-2}$, this is a common default value for MT solvers for 1D PDEs such as MOVCOL [32].

We see that the MMPDE method can be used effectively as an adaptation approach for solving PDEs when using a heuristic monitor function based on solution information at previous time steps, in this case, the solution arc-length. In the next section, we discuss some preliminary examples using an error estimation-based monitor function, with some discussion of more general error estimation-based MT algorithms.

3.3 Moving Transformation Approach for PDEs Using an Error Estimate Based Monitor Function

As one of the objectives of this thesis is to explore MT methods which could possibly be applied in the context of adaptive error control algorithms, this section proceeds in this direction by exploring some preliminary approaches for error estimation-based MT adaptation. Additionally, to aid in future work in the development of error control algorithms based on MT adaptivity, an adaptive error control algorithm is proposed which provides spatial and temporal adaptivity for 1D time-dependent PDEs.

To apply a simple MT method based on a monitor function which is in turn based on an error estimate, using the second-order finite difference method implemented in Section 3.2, we make use of a monitor function of the form

$$M(x,t) = (\epsilon_{MACH} + E(x,t))^{\frac{1}{p}}, \tag{3.15}$$

where p is the order of the spatial discretization method and $E(x,t)$ is an error estimate for the approximate solution at time t. For our experiments, this error estimate is of the form

$$E(x(\xi,t),t) = |U(x(\xi,t),t) - \widetilde{U}(x(\xi,t),t)|, \tag{3.16}$$

where U is the current solution approximation and $\widetilde{U}$ is another solution approximation having a higher order of accuracy. Here we generate $\widetilde{U}$ using a standard fourth order finite difference discretization. For a uniform mesh $\{x_i\}_{i=1}^{N+1}$ of N subintervals having subinterval length h we apply the centered formulas [21]

$$\widetilde{U}_x(x_i,t) \approx \frac{1}{h}\Big(\frac{1}{12}\widetilde{U}(x_{i-2},t) - \frac{2}{3}\widetilde{U}(x_{i-1},t) + \frac{2}{3}\widetilde{U}(x_{i+1},t) - \frac{1}{12}\widetilde{U}(x_{i+2},t)\Big), \tag{3.17}$$

$$\begin{aligned}\widetilde{U}_{xx}(x_i,t) \approx \frac{1}{h^2}\Big(&-\frac{1}{12}\widetilde{U}(x_{i-2},t) + \frac{4}{3}\widetilde{U}(x_{i-1},t) - \frac{5}{2}\widetilde{U}(x_i,t)\\ &+\frac{4}{3}\widetilde{U}(x_{i+1},t) - \frac{1}{12}\widetilde{U}(x_{i+2},t)\Big),\end{aligned} \tag{3.18}$$

at the points $x_i, i = 3, ..., N-1$. At the points x_1 and x_2, we apply the non-symmetric schemes

$$\widetilde{U}_x(x_1,t) \approx \frac{1}{h}\left(-\frac{25}{12}\widetilde{U}(x_1,t) + 4\widetilde{U}(x_2,t) - 3\widetilde{U}(x_3,t) + \frac{4}{3}\widetilde{U}(x_4,t) - \frac{1}{4}\widetilde{U}(x_5,t)\right), \tag{3.19}$$

$$\widetilde{U}_{xx}(x_1,t) \approx \frac{1}{h^2}\left(\frac{15}{4}\widetilde{U}(x_1,t) - \frac{77}{6}\widetilde{U}(x_2,t) + \frac{107}{6}\widetilde{U}(x_3,t) - 13\widetilde{U}(x_4,t) + \frac{61}{12}\widetilde{U}(x_5,t) - \frac{5}{6}\widetilde{U}(x_6,t)\right), \tag{3.20}$$

$$\widetilde{U}_x(x_2,t) \approx \frac{1}{h}\left(-\frac{1}{4}\widetilde{U}(x_1,t) - \frac{5}{6}\widetilde{U}(x_2,t) + \frac{3}{2}\widetilde{U}(x_3,t) - \frac{1}{2}\widetilde{U}(x_4,t) + \frac{1}{12}\widetilde{U}(x_5,t)\right), \tag{3.21}$$

$$\widetilde{U}_{xx}(x_2,t) \approx \frac{1}{h^2}\left(\frac{5}{6}\widetilde{U}(x_1,t) - \frac{5}{4}\widetilde{U}(x_2,t) - \frac{1}{3}\widetilde{U}(x_3,t) - \frac{7}{6}\widetilde{U}(x_4,t) - \frac{1}{2}\widetilde{U}(x_5,t) + \frac{1}{12}\widetilde{U}(x_6,t)\right), \tag{3.22}$$

and at $i = N$ we apply the backward finite difference schemes associated with (3.21-3.22) and at $i = N + 1$ we apply the backward schemes associated with (3.19-3.20). The lower order solution U and higher order solution $\widetilde{U}$ are computed simultaneously along with MMPDE5, allowing the error estimation monitor function to be updated after each successive time step.

As discussed previously, we must be able to initially adapt the coordinate transformation based on some initial solution information. For the purposes of this experimentation, we wish for this initial adaptation to be based on an error estimate, though it could also be obtained through other means such as deriving it from the initial condition. To obtain the initial monitor function $M(x, t_0)$, defined as in (3.16), we first solve U and $\widetilde{U}$ on a uniform mesh in Ω_p for a small time interval. These initial solution approximations are then used to produce the initial error estimate for use in generating $M(x, t_0)$, allowing for an initial coordinate transformation which is well-adapted to regions of difficulty as indicated by the solution error. For $N = 40$, $\tau = 10^{-2}$ and 5 iterations of the monitor function smoothing, the solution to OLBE with $\epsilon = 10^{-2}$ was computed and plotted in Figure 3.7. Figure 3.8 shows TLBE with $\epsilon = 10^{-2}$ solved in the same way. In both of these cases, we see superior performance from the MT method using this error estimation-based MT adaptation as opposed to applying the same discretization on a uniform mesh in Ω_p. For the solution to OLBEϵ2 plotted in 3.7, the solution error at final time $t = 1$ is 0.0058, an improvement compared to the solution obtained by discretizing the problem directly using a uniform mesh on Ω_p, which produces a solution with error 0.0535. The solution to TLBEϵ2 in Figure 3.8 has error 0.0052, an improvement over the uniform mesh case which produces a solution with error 0.0370.

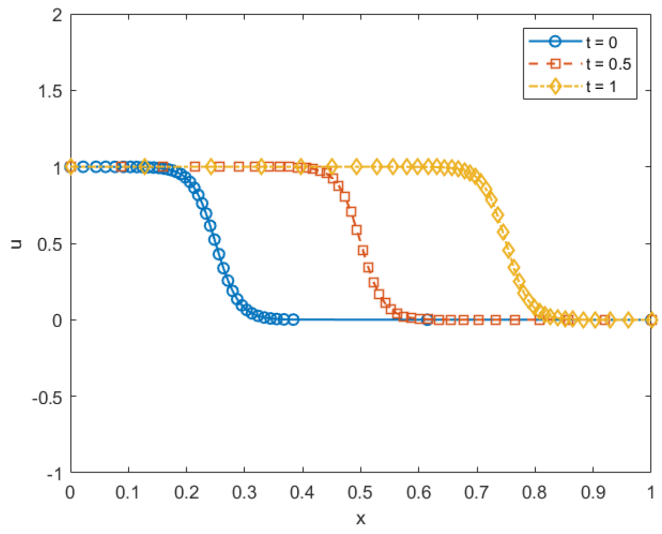


Figure 3.7: OLBEϵ2. Solved using finite difference discretization (1.7)-(1.9) with $N = 40$ using a MT method with error estimation monitor function.

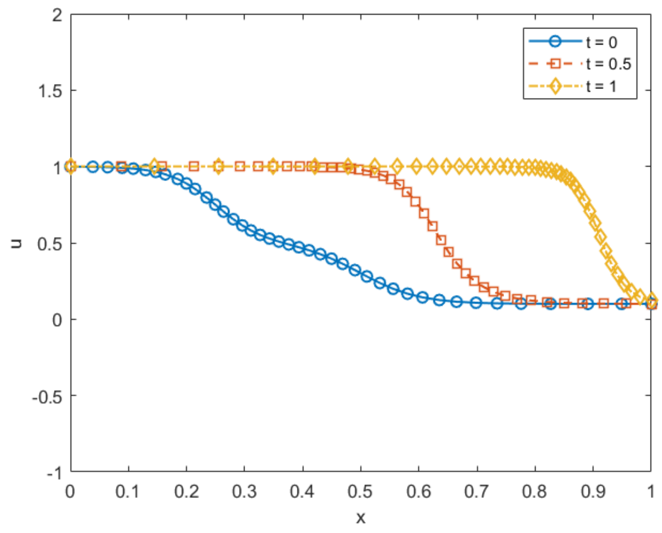


Figure 3.8: TLBEϵ2. Solved using finite difference discretization (1.7)-(1.9) with $N = 40$ using a MT method with error estimation monitor function.

An obvious issue when applying this monitor function, and in general applying monitor functions based on error estimation in the PDE case, is the fact that the adaptivity will most effective when the error estimate is large in some area. However, this means that in order to have effective adaptation, we must have that error has already been introduced into the approximate solution. Unless this error is addressed as it occurs, depending on how large it is, it can propagate through the solution as the PDE is integrated further in time, potentially leading to catastrophic errors. This was of little concern in the BVODE case, as for these problems the whole solution is regenerated each time the solution was to be improved, but in the PDE case, the quality of the solution at any point in time is dependent on the quality of the solution at previous times. BACOLI addresses this by accepting a solution after a time step s, $\underline{U}(x, t + s)$ only if the estimated error is within the user provided tolerances. If the tolerances are not met, then the solution at this step is rejected, and the solution reverts back to the solution from the previous time step, $\underline{U}(x, t)$. The mesh is then adapted based on the size and distribution of the estimated error in $\underline{U}(x, t + s)$ and the step is attempted again. This process is repeated until a solution which has error satisfying the tolerances is obtained. This kind of iterative refinement procedure will be necessary for an error control algorithm based on MT methods to function. This procedure must also involve the addition and removal of mesh points when required, which allows the computation to adapt to the solution behaviour sufficiently well to meet the tolerance at points in time where the problem is difficult, as well as to operate efficiently by using the minimum required number of points at points in time where the problem is less difficult. This addition and removal of mesh points has not been well-examined in the literature on MT methods. A description of a possible approach to error control implementing MT methods is described in the following section.

3.4 Outline of an Adaptive Error Control Algorithm using Moving Transformation Methods

Since we have seen that an iterative adaptive error control procedure is required in order to properly implement error-based adaptation for PDEs using an MT algorithm, this section discusses a possible approach for an error control MT algorithm for 1D time-dependent PDEs with adaptation driven through estimates of the solution error. This algorithm is intended to be a starting point for future work, primarily providing a description of how an error control MT method could work in theory, and we also hope that similar approaches could be generalized to higher dimensional problems. We avoid implementation-specific details such as the choice of error estimation-based monitor function and choice of MMPDE, as decisions such as these will likely require further investigation. We assume the use of a continuous solution approximation $\underline{U}(x,t)$. Error estimates are assumed to be derived using a solution approximation $\widetilde{\underline{U}}(x,t)$ which is of higher order than the approximate solution $\underline{U}(x,t)$. Methods for obtaining an error estimation monitor function in this context are not discussed here, but could likely be based on interpolation-based error estimators such as those implemented in BACOLI.

For this section, we also consider an alternative perspective on the MT methods, which instead of considering coordinate transformations mapping the computational domain Ω_c to physical domain Ω_p, makes use of a mapping from Ω_p to Ω_c. MT methods implementing these approaches have been applied successfully to higher-dimensional PDE problems and offer certain advantages over the more standard formulation [4]. One advantage of using such a mapping is that the PDE can be transformed to Ω_c, and its transformed solution defined on this domain, but this solution can be evaluated as if it was defined on Ω_p without requiring any interpolation between the domains.

In this proposed algorithm, we wish to decouple the computation of the MMPDE which generates the coordinate transformation (the solution to the MMPDE) from the computation of the solution to the transformed physical PDE. This allows us to better control the adaptivity by forcing the coordinate transformation to always be well-adapted to the monitor function at the current time and additionally removes the non-linear coupling associated with solving the MMPDE and the transformed physical PDE together, allowing both the MMPDE and the transformed physical PDE to be solved efficiently.

As mentioned above, instead of using an MMPDE which has a solution $x(\xi,t)$ which maps the smoothed computational space Ω_c to the physical domain Ω_p, we consider MMPDEs whose solution is the inverse function $\xi(x,t) : \Omega_p \mapsto \Omega_c$. An example of an MMPDE generating this kind of coordinate transformation is MMPDE5xi, a modification of MMPDE5, given by [4]

$$\xi_t(x,t) = \frac{1}{\tau}\left(\frac{1}{M(x,t)}\xi_x(x,t)\right)_x. \tag{3.23}$$

Note that MMPDExi is defined on Ω_p, the physical domain. For the purposes of describing this algorithm, we use MMPDE5xi, though it is unclear which MMPDE would be best in a practical implementation. (We consider MMPDE5xi for the generation of the coordinate transformation $\xi(x,t)$; however, it is of course possible that the corresponding inverse coordinate transformation $x(\xi,t)$ could be used in order to implement the more standard MT approaches as discussed in previous sections). This MT approach will be a rezoning algorithm with spatial discretization done in Ω_c. Similar to the case where we derived MT methods for PDEs in terms of the transformation $x(\xi,t)$, we define the transformed solution on Ω_c by

$$\underline{\hat{u}}(\xi(x,t),t) = \underline{u}(x,t). \tag{3.24}$$

Using MMPDE5xi, we then have the following coupled PDE system

$$\begin{aligned}\underline{\hat{u}}_t(\xi(x,t),t) &= f\Big(t, x, \underline{\hat{u}}(\xi(x,t),t), \underline{\hat{u}}_\xi(\xi(x,t),t)\xi_x, \underline{\hat{u}}_{\xi\xi}(\xi(x,t),t)\xi_x^2 + \underline{\hat{u}}_\xi(\xi(x,t),t)\xi_{xx}\Big) + \underline{\hat{u}}_\xi(\xi(x,t),t)\xi_x\xi_t \\ \xi_t(x,t) &= \frac{1}{\tau}\left(\frac{1}{M(x,t)}\xi_x(x,t)\right)_x,\end{aligned} \tag{3.25}$$

with boundary conditions

$$\underline{b}_L(t, \underline{\hat{u}}(0,t), \underline{\hat{u}}_\xi(0,t) = \underline{0}, \tag{3.26}$$

$$\underline{b}_R(t, \underline{\hat{u}}(1,t), \underline{\hat{u}}_\xi(1,t) = \underline{0}, \tag{3.27}$$

and the initial conditions

$$\underline{\hat{u}}(\xi(x,t),t) = \underline{u}_0(x). \tag{3.28}$$

As we consider the movement of $\xi(x,t)$ to occur independently of the evolution of the solution to the physical PDE through time, the system (3.25) is decoupled during the computation. To ensure that $\xi(x,t)$ is sufficiently well-adapted to the solution initially, MMPDE5xi is integrated through virtual time until it is sufficiently adapted to the initial monitor function, and $\underline{\hat{u}}(\xi(x,t),t)$ is then solved for one time step by discretizing it with an appropriate number of subintervals of a uniform computational mesh on Ω_c. Since we consider the coordinate transformation to be fixed in time during the discretization of the transformed physical PDE, in (3.25) we can set $\xi_t(x,t) = 0$, giving us a simplified form of the transformed physical PDE on Ω_c

$$\underline{\hat{u}}_t(\xi(x,t),t) = f\Big(t, x, \underline{\hat{u}}(\xi(x,t),t), \underline{\hat{u}}_\xi(\xi(x,t),t)\xi_x, \underline{\hat{u}}_{\xi\xi}(\xi(x,t),t)\xi_x^2 + \underline{\hat{u}}_\xi(\xi(x,t),t)\xi_{xx}\Big). \tag{3.29}$$

In this algorithm, we consider $M(x,t)$ to be derived from the estimated error of a continuous approximate solution $\underline{U}(x,t)$, with $\underline{\hat{U}}(\xi(x,t),t)$ being its analog on Ω_c. To demonstrate how this might work, we assume that at time t_i we have the current solution approximation on Ω_c, $\hat{U}(\xi(x,t_i),t_i), t_i > t_0$, defined in terms of the current coordinate transformation $\xi(x,t_i)$. We note that in practice, $\xi(x,t_i)$ is not, strictly speaking, the value of $\xi(x,t_i)$, but rather has been obtained through virtual time integration such that it is appropriate for the solution behaviour at time t_i.

We wish to obtain the solution to the transformed physical PDE - and hence the solution to the physical PDE - at time $t_{i+1} > t_i$. Using an error estimation-based monitor function associated with the current solution approximation on Ω_p, $\underline{U}(x,t_i) \equiv \underline{\hat{U}}(\xi(x,t_i),t_i)$, the MMPDE is integrated through virtual time to obtain a coordinate transformation which is better adapted to the solution at time t_{i+1}, $\xi(x,t_{i+1})$. In discretizing the physical PDE appearing in (3.29), we wish to transform the PDE to Ω_c using $\xi(x, t_{i+1})$ so that it can be discretized accurately using a uniform mesh of N subintervals, $\{\xi_i\}_{j=1}^{N+1} \subset \Omega_c$. However, the current solution at t_i has been defined in terms of the previous coordinate transformation $\xi(x, t_i)$, and we require that the transformed physical PDE be discretized in terms of $\xi(x, t_{i+1})$ so that its solution is appropriately smoothed on Ω_c. Therefore we purpose an interpolation procedure between these two coordinate transformations such that the PDE can be expressed in terms of $\underline{\hat{U}}(\xi(x, t_{i+1}), t)$ using the known solution information from the previously computed solution $\underline{\hat{U}}(\xi(x, t_i), t_i)$. This process may involve the inverse interpolation of $\xi(x, t_{i+1})$ to obtain its inverse coordinate transformation $x(\xi, t_{i+1})$ so that the values $\{x_j = x(\xi_j, t_{i+1})\}_{j=1}^{N+1}$ can be obtained. In this way, the points required for the discretization of $\underline{\hat{U}}(\xi(x, t_{i+1}), t_i)$, $\{\underline{\hat{U}}(\xi(x_j, t_{i+1}), t_i)\}_{j=1}^{N+1}$, as well as its derivatives, can be recovered. Therefore (3.29) can be discretized in terms of the updated coordinate transformation and then integrated forward to time t_{i+1} to obtain the solution $U(\xi(x, t_{i+1}), t_{i+1})$.

This solution procedure can be repeated on the current time step until a solution with an estimated error satisfying a user tolerance is obtained. The accuracy can be increased by further adapting the coordinate transformation and also by updating the number of points, N, used in the uniform computational mesh. The error estimate on each step is also used to generate the monitor function so that the algorithm fully controls the solution error. The transformation can be updated on each time step, or alternatively, when the error estimate is met on a given time step, it can be fixed until the next time the error is above the tolerance, avoiding the complicated inverse interpolation step and increasing efficiency. This algorithm is given a complete description in Algorithm 6, where we assume that the coordinate transformation is updated each time the physical PDE is to be solved.

Algorithm 6 provides a general method for generating continuous, error-controlled solutions to 1D PDEs, though many of the more complicated algorithmic details which will have to be considered in a practical implementation are left out. Implementation specific details could be based on the approaches used in BACOLI, such as interpolation based spatial error estimation to provide error control as well as drive adaptivity. For the choice of MMPDE used in the implementation, to manage

Algorithm 6: Error Control MT PDE algorithm.

```
1  function ErrorControlMT ($N_0$, $x_a$, $x_b$, $t_0$, $t_{out}$, $PDE$, $MMPDE$, $ATOL$, $RTOL$);
   Input  : Initial number of mesh subintervals $N_0$; spatial boundary points $x_a, x_b$; starting time $t_0$;
            output time $t_{out}$; problem definition $PDE$; moving mesh PDE $MMPDE$; absolute and
            relative error tolerances for time-stepping $ATOL$ and $RTOL$
   Output: Approximate solution $U(x, t_{out})$.
2  // Generate initial uniform mesh on $\Omega_p$
3  $\{x_i\}_{i=1}^{N_0+1}$ := linspace($x_a$, $x_b$, $N_0 + 1$)
4  // Solve the PDE forward for a small amount of time $t = t_0 + \delta$ using uniform mesh
5  // to get approximate solution $U$ and higher order approximation $\widetilde{U}$
6  $[U, \widetilde{U}] := SolvePDE(\{x_i\}_{i=1}^{N_0+1}, t_0 + \delta, PDE(x))$
7  // Take initial error estimate.
8  $Err := Errest(U, \widetilde{U})$
9  // Generate the initial error estimation monitor function.
10 $M := MonitorGen(Err)$
11 // Solve the MMPDE with this fixed monitor function until convergence criteria is met.
12 $\xi^{(t_0)}(x) := SolveMMPDE(\{x_i\}_{i=1}^{N_0+1}, M)$
13 $t := t_0$
14 // Using the global error estimate to decide the number of subintervals.
15 $N := DecideN(N.GE)$
16 // Map the PDE to the computational domain.
17 $[\hat{U}, \bar{U}] := InterpolateOn([U, \widetilde{U}], \xi^{(0)}(x))$
18 // Begin the solution procedure.
19 while $t < t_{out}$ do
20     // Compute solution at next time step $s$.
21     $[U, \widetilde{U}, s] := SolvePDE(\{\xi_i\}_{i=1}^{N_0+1}, t, PDE(\xi^{(t)}(x)))$
22     // Take the error estimate.
23     $Err := Errest(\hat{U}, \bar{U})$
24     // Generate the initial error estimation monitor function.
25     $M := MonitorGen(Err)$
26     while not TOLMet(Err, ATOL, RTOL) do
27         // Update the transformation.
28         $\xi^{(t+s)}(x) = SolveMMPDE(\{x_i\}_{i=1}^{N+1}, \mathrm{M})$
29         // Generate new $N$ value, number of points to use on the uniform computational mesh.
30         $N := DecideN(N, Err)$
31         // Interpolate solution on new transformation.
32         $[\hat{U}, \bar{U}] := InterpolateOn([\hat{U}, \bar{U}], \xi^{(t+s)}(x))$
33         // Attempt to take the step again.
34         $[U, \widetilde{U}, s] := SolvePDE(\{\xi_i\}_{i=1}^{N_0+1}, t, PDE(\xi^{(t+s)})(x))$
35         // Take the error estimate.
36         $Err := Errest(\hat{U}, \bar{U})$
37     end
38     // After a step is accepted, for efficiency purposes, $N$ can be increased or reduced.
39     $N := DecideN(N, Err)$
40     $t = t + s$
41 end
42 return $U(x, t_{out})$
```

the effects of spatial smoothing on the effectiveness and efficiency of the algorithm without relying on discrete smoothing of the monitor function, implementation of MMPDEs which include spatial smoothing in their formulation may be ideal. MMPDEs with spatial smoothing are analyzed in [34] and are implemented in the MOVCOL algorithm, which the authors report to be a robust algorithm for general 1D PDE problems [32]. The spatially smoothed MMPDE which is implemented in MOVCOL is given by

$$\tau\Big(I - \gamma\partial_{\xi\xi}\Big)x_t(\xi, t) = -\Big(M(x(\xi, t), t)x_\xi(\xi, t)\Big)_\xi, \tag{3.30}$$

where I is the identity operator and $(I - \gamma\partial_{\xi\xi})$ is a spatial smoothing operator. The parameter γ gives control over the spatial smoothing, analogous to the local smoothing applied previously, and the parameter τ governs temporal relaxation of the MMPDE as in MMPDE5. The use of this equation with τ and γ chosen appropriately has the potential to be very effective and removes the requirement of smoothing discrete evaluations of the monitor function.

We include implementations of this algorithm, particularly as a modification to BACOLI, as a topic for future work. It is possible that this algorithm will be suitable for 2D problems as well.

3.5 Moving Transformation Approaches in 2D

As we have mentioned previously, MT methods are of principal interest in the context of PDE problems having two or more spatial dimensions. As we saw in Chapter 1, simple approximations such as de Boor's algorithm provide sufficiently accurate and efficient approximations to an equidistributed mesh. Such a simple means of generating well-adapted grids for 2D problems are not available for reasons we will discuss in this section. For the remainder of this chapter, we describe some basics of MT adaptation for time-dependent PDE problems having 2 spatial dimensions.

For 2D PDEs we require a map $\boldsymbol{x}(\boldsymbol{\xi}, t)$ between compact sets $\{(\xi, \eta)^T : \xi, \eta \in \mathbb{R}\} = \Omega_c$ and $\{(x, y)^T : x, y \in \mathbb{R}\} = \Omega_p$. With no loss in generality, assume $\Omega_c = \Omega_p = [0, 1] \times [0, 1]$. As in the

1D case, Ω_c is considered to be a smoothed computational domain and Ω_p the physical domain on which the PDE is defined, with $\boldsymbol{x}(\boldsymbol{\xi}, t) = (x(\xi, \eta, t), y(\xi, \eta, t))^T$ mapping the computational variables onto Ω_p. For the coordinate transformation $\boldsymbol{x}(\boldsymbol{\xi}, t)$ to effectively transform the physical PDE such that it can be solved on a uniform mesh in Ω_c, the approach in 1D is emulated, requiring that $\boldsymbol{x}(\boldsymbol{\xi}, t)$ equidistribute a monitor function $M(\boldsymbol{x}(\boldsymbol{\xi}, t), t)$. See [4] for an in-depth discussion on monitor functions for 2D problems.

In 2D an equidistributing coordinate transformation is one which satisfies the condition [5, 3]

$$M(\boldsymbol{x}(\boldsymbol{\xi}, t), t)|\boldsymbol{J}(\boldsymbol{\xi}, t)| = \theta(t), \qquad \theta(t) = \int_{\Omega_p} M(\boldsymbol{x}(\boldsymbol{\xi}, t), t) d\boldsymbol{x}, \tag{3.31}$$

with $\boldsymbol{J}(\boldsymbol{\xi}, t)$ being the Jacobian matrix

$$J = \begin{bmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{bmatrix}. \tag{3.32}$$

For a rectangular computational grid on Ω_c of dimension $N \times M$, this condition specifies that the volume (in terms of the double integral of the monitor function) enclosed in the mapping of each sub-rectangle to Ω_p must be equal. However, there are uncountably infinitely many shapes that the image of each rectangle can take while still satisfying this condition, meaning additional constraints must be imposed in order for $\boldsymbol{x}(\boldsymbol{\xi}, t)$ to be unique for a given monitor function. These additional constraints can be based on many different properties that we desire $\boldsymbol{x}(\boldsymbol{\xi}, t)$ to satisfy, but here we limit our discussion to an interesting class of 2D MMPDEs which are based on the theory of optimal transport.

Use of optimal transport based MMPDEs is motivated by some problems which are common in mesh generation algorithms [3]. These problems are mesh tangling, where over time grid points begin to cross each other causing catastrophic error [3]. In 1D, this corresponds to a loss in monotonicity of the transformation. Additionally, these MMPDEs seek to generate meshes which do not have include sub-regions having long, thin corners [3]. The process of discretizing a differential equation on such a mesh is known to be poorly conditioned. These issues can be alleviated by prescribing that the coordinate transformation satisfy both the equidistribution condition (3.27) and minimize the functional (least squares norm)

$$\int_{\Omega_p} |\boldsymbol{x}(\boldsymbol{\xi}, t) - \boldsymbol{\xi}|^2 d\boldsymbol{\xi}. \tag{3.33}$$

Note that this condition corresponds to stating that the optimal transformation is the equidistributing transformation which maps points in Ω_c to the nearest possible points in Ω_p and which still permits the equidistribution condition to hold. That is, we seek the unique coordinate transformation that is closest to the identity transformation $\boldsymbol{x}(\boldsymbol{\xi}, t) = \boldsymbol{\xi}$. These conditions are sufficient to specify a unique coordinate transformation, which is obtained as the gradient of a convex mesh functional $P(\boldsymbol{\xi}, t)$,

$$\boldsymbol{x}(\boldsymbol{\xi}, t) = \nabla_{\boldsymbol{\xi}} P(\boldsymbol{\xi}, t), \tag{3.34}$$

where $\nabla_{\boldsymbol{\xi}} = (\partial_\xi, \partial_\eta)^T$. $P(\boldsymbol{\xi}, t)$ is obtained as the solution to the Monge-Ampére equation [3]

$$M(\nabla_{\boldsymbol{\xi}} P(\boldsymbol{\xi}, t), t)|\boldsymbol{H}(P(\boldsymbol{\xi}, t))| = \theta(t), \tag{3.35}$$

where $\boldsymbol{H}(P(\boldsymbol{\xi}, t))$ is the Hessian matrix of $P(\boldsymbol{\xi}, t)$. This equation can be solved for $P(\boldsymbol{\xi}, t)$ when combined with appropriate boundary conditions [3].

As when generalizing from MMPDE0 to MMPDE5, a temporally relaxed version of (3.35) is often implemented in practical computation. The Parabolic Monge-Amphére MMPDE is a temporally relaxed analog to (3.31), given by

$$\tau\left(1-\gamma\nabla_{\boldsymbol{\xi}}\right)Q_t(\boldsymbol{\xi},t) = \left(M(\nabla_{\boldsymbol{\xi}}Q(\boldsymbol{\xi},t))|\boldsymbol{H}(Q(\boldsymbol{\xi},t))|\right)^{\frac{1}{2}},$$
$$\boldsymbol{x}(\boldsymbol{\xi},t) = \nabla_{\boldsymbol{\xi}}Q(\boldsymbol{\xi},t), \quad \tau > 0, \quad \gamma > 0, \tag{3.36}$$

where $Q(\boldsymbol{\xi},t)$ is a convex mesh functional, τ governs temporal relaxation and γ governs spatial smoothing [5]. This equation has been effectively used within MT methods for solving difficult high-dimensional PDE problems, such as those appearing in weather modelling [3].

An MMPDE such as the Parabolic Monge-Amphére equation can be coupled to a physical 2D PDE in the same way as was done in the 1D case; since the resulting expressions are quite complicated, we do not give them here. This likely means that similar algorithms can often be applied in both cases. However, due to the size of 2D problems and the highly non-linear nature of the simultaneous approach, the alternating and rezoning approaches are typically implemented in this case. Ideally, algorithms such as the error control algorithm discussed in the previous section could be used to provide fully error-controlled solutions to 2D PDEs. As mentioned earlier, a possibility for the application of these 2D MT methods is combining them with the 2D tensor product B-spline Gaussian collocation algorithm implemented in BACOL2D, improving the effectiveness of this solver for problems exhibiting sharply varying spatial structures.

Chapter 4

Conclusions & Future Work

4.1 Conclusions

In this thesis, we have seen that MT methods are capable of effectively adapting the solution to BVODEs and 1D PDEs. Additionally, we have seen in both cases that this MT adaptation can be based on error-estimation, leading to the potential for implementation of MT methods in adaptive error control algorithms. Computational experiments revealed many important facets of MT methods such as choice and smoothness of the monitor function, MT equation choice, as well as parameter choices for the MT equation that will have to be further developed and analyzed in order for the methods to be applied in a full-scale implementation. An error control algorithm using a rezoning procedure with spatial discretization on a uniform computational mesh was discussed which could possibly serve as a basis for future investigation of this approach.

Throughout this work, we have carefully chosen the wording we use in order to distinguish MT approaches from traditional moving mesh approaches. Typically, approaches implementing MT methods in the way we discuss are also categorized as moving mesh methods, which we believe obfuscates the significance of MT methods, where the only actual meshes involved are uniform, static meshes in a computational domain. The adaptivity in MT methods is driven through a coordinate transformation obtained as the solution to a "moving mesh" differential equation, which provides an analogy between MT methods and traditional moving mesh (i.e., traditional r-adaptivity) methods. However, this analogy only truly holds when the coordinate transformation is used to map a uniform mesh in Ω_c back onto a non-uniform mesh in Ω_p. In this thesis we have used the coordinate transformation to transform the physical PDE on Ω_p to a transformed PDE on Ω_c which has a smoother solution. While the two approaches are almost equivalent in a fully mathematical setting, in practice, there are many important distinctions, enough so that we believe the more traditional moving mesh approaches and MT approaches benefit from being treated as distinct but related concepts.

4.2 Future Work

There are many possible avenues for future work. One possibility is the development of error control software for 1D PDEs, possibly written as extensions to existing software such as BACOLI. Another natural piece of future work is the implementation of MT methods in general-purpose adaptive software for 2D PDEs. For the 1D case, MOVCOL demonstrates that some of the important components of the MT methods, namely the adaptivity through solving an MMPDE, can be applied in a robust solver; however, this is not an MT solver in the way we define the methods here. It is an example of what we have referred to as a traditional r-adaptivity solver; the coordinate transformation is used to obtain a non-uniform mesh on Ω_p.

In the 2D PDE case, to our knowledge, there are no general-purpose solvers implementing the MT methods, or even a partial implementation such as MOVCOL. One direction for work of this type could involve modification to the BACOL2D code to implement this kind of adaptation, allowing its high-order tensor product B-spline Gaussian collocation algorithm to be used in conjunction with MT based spatial adaptivity, likely improving its performance on problems exhibiting sharp spatial features which often occur in the modelling of phenomena such as fluids. Work following this would include refinement of the BACOL2D algorithm to include interpolation-based spatial error estimation and an iterative solution procedure to provide adaptive error control. An important factor in most of this work would include the development of efficient discretization algorithms for solving the chosen MMPDE which would preserve important properties such as monotonicity.

Bibliography

[1] T. Chan and J. Shen, *Image processing and analysis: variational, PDE, wavelet, and stochastic methods.* Society for Industrial and Applied Mathematics, 2005.

[2] B. Keyfitz and N. Keyfitz, "The McKendrick partial differential equation and its uses in epidemiology and population study," *Mathematical and Computer Modelling*, vol. 26, no. 6, pp. 1–9, 1997.

[3] E. Walsh, *Moving mesh methods for problems in meteorology.* PhD thesis, University of Bath, 2010.

[4] W. Huang and R. Russell, *Adaptive moving mesh methods.* Springer, 2010.

[5] C. Budd, W. Huang, and R. Russell, "Adaptivity with moving grids," *Acta Numerica*, vol. 18, p. 111–241, 2009.

[6] G. Bader and U. Ascher, "A new basis implementation for a mixed order boundary value ODE solver," *SIAM Journal on Scientific and Statistical Computing*, vol. 8, no. 4, pp. 483–500, 1987.

[7] J. Pew, Z. Li, and P. Muir, "Algorithm 962: BACOLI: B-spline adaptive collocation software for PDEs with interpolation-based spatial error control," *ACM Transactions on Mathematical Software*, vol. 42, no. 3, p. 25, 2016.

[8] W. Huang and R. Russell, "A moving collocation method for solving time dependent partial differential equations," *Applied Numerical Mathematics*, vol. 20, no. 1, pp. 101–116, 1996.

[9] Z. Li and P. Muir, “B-spline Gaussian collocation software for two-dimensional parabolic PDEs,” *Advances in Applied Mathematics and Mechanics*, vol. 5, no. 4, pp. 528–547, 2013.

[10] U. Ascher, R. Mattheij, and R. Russell, *Numerical solution of boundary value problems for ordinary differential equations.* Society for Industrial and Applied Mathematics, 1994.

[11] “Scilab, *bvode*.” Scilab Enterprises, 143 bis rue Yves Le Coz, 78000 Versialles, France.

[12] “scipy.integrate.solve_bvp.” SciPy v1.1.0 Reference Guide.

[13] R. Burden and D. Faires, *Numerical Analysis.* Brooks/Cole, 2011.

[14] P. Muir and B. Owren, “Order barriers and characterizations for continuous mono-implicit Runge-Kutta schemes,” *Mathematics of Computation*, vol. 61, no. 204, pp. 675–699, 1993.

[15] L. Shampine, P. Muir, and H. Xu, “A user-friendly fortran BVP solver,” *Journal of Numerical Analysis, Industrial and Applied Mathematics*, vol. 1, no. 2, pp. 201–217, 2006.

[16] M. Adams, C. Tannahill, and P. Muir, “Error control Gaussian collocation software for boundary value ODEs and 1D time-dependent PDEs,” *to appear in Numerical Algorithms*, 2019.

[17] W. Enright and P. Muir, “Superconvergent interpolants for the collocation solution of boundary value ordinary differential equations,” *SIAM Journal on Scientific Computing*, vol. 21, no. 1, pp. 227–254, 1999.

[18] R. Wang, P. Keast, and P. Muir, “BACOL: B-spline adaptive collocation software for 1-D parabolic PDEs,” *ACM Transactions on Mathematical Software*, vol. 30, no. 4, pp. 454–470, 2004.

[19] R. Wang, P. Keast, and P. Muir, “Algorithm 874: BACOLR—spatial and temporal error control software for PDEs based on high-order adaptive collocation,” *ACM Transactions on Mathematical Software*, vol. 34, no. 3, p. 15, 2008.

[20] M. S. Gockenbach, *Partial differential equations: analytical and numerical methods.* Society for Industrial and Applied Mathematics, 2005.

[21] B. Fornberg, "Generation of finite difference formulas on arbitrarily spaced grids," *Mathematics of Computation*, vol. 51, no. 184, pp. 699–706, 1988.

[22] L. Petzold, "Description of DASSL: a differential/algebraic system solver," tech. rep., Sandia National Labs., Livermore, CA (USA), 1982.

[23] E. Hairer and G. Wanner, "RADAU5-an implicit Runge-Kutta code," *Report, Université de Geneve, Dept. de Mathématiques, Geneve*, 1988.

[24] "dae." Scilab Enterprises, 143 bis rue Yves Le Coz, 78000 Versailles, France.

[25] L. F. Shampine, M. W. Reichelt, and J. A. Kierzenka, "Solving index-1 DAEs in MATLAB and Simulink," *SIAM Review*, vol. 41, no. 3, pp. 538–552, 1999.

[26] C. De Boor, "Package for calculating with B-splines," *SIAM Journal on Numerical Analysis*, vol. 14, no. 3, pp. 441–472, 1977.

[27] T. Arsenault, T. Smith, P. Muir, and P. Keast, "Efficient interpolation based error estimation for 1D time-dependent PDE collocation codes," *Technical Report 2011_001, Department of Mathematics and Computing Science, Saint Mary's University, Halifax, NS*, 2011.

[28] R. Wang, P. Keast, and P. Muir, "A high-order global spatially adaptive collocation method for 1-D parabolic PDEs," *Applied Numerical Mathematics*, vol. 50, no. 2, pp. 239–260, 2004.

[29] J. Pew, Z. Li, C. Tannahill, P. Muir, and G. Fairweather, "Performance analysis of error-control B-spline Gaussian collocation software for PDEs," *Computers & Mathematics with Applications*, 2018.

[30] C. Tannahill and P. Muir, "Application of error control software to ODE and PDE-based epidemiological models," *Technical Report 2017_001, Department of Mathematics and Computing Science, Saint Mary's University, Halifax, NS*, 2017.

[31] C. de Boor, "Good approximation by splines with variable knots," in *Spline functions and approximation theory*, pp. 57–72, Springer, 1973.

[32] W. Huang and R. Russell, "A moving collocation method for solving time dependent partial differential equations," *Applied Numerical Mathematics*, vol. 20, no. 1, pp. 101–116, 1996.

[33] F. Fritsch and R. Carlson, "Monotone piecewise cubic interpolation," *SIAM Journal on Numerical Analysis*, vol. 17, no. 2, pp. 238–246, 1980.

[34] W. Huang and R. Russell, "Analysis of moving mesh partial differential equations with spatial smoothing," *SIAM Journal on Numerical Analysis*, vol. 34, no. 3, pp. 1106–1126, 1997.

[35] J. Boisvert, P. Muir, and R. Spiteri, "A Runge-Kutta BVODE solver with Global Error and Defect Control," *ACM Transactions on Mathematical Software*, vol. 39, no. 2, pp. 11:1–11:22, 2013.