Finite Difference Based Error Estimation for Boundary Value ODEs, 1D PDEs and 2D PDEs

By
Thomas Murtha

A Thesis Submitted to
Saint Marys University, Halifax, Nova Scotia
in Partial Fulfillment of the Requirements for
the Degree of B.Sc. Mathematics and Computing Science

August, 2017, Halifax, Nova Scotia

Approved: Dr. Paul Muir

Approved: Dr. Wendy Finbow-Singh

Date: August 26, 2017

Finite Difference Based Error Estimation for Boundary Value ODEs, 1D PDEs and 2D PDEs

Tom Murtha

August 26, 2017

Abstract

Differential equations are involved in many fields. Often these cannot be solved exactly, so they must be approximated. To do this, we derive a fourth order finite difference scheme for non-uniform meshes. This is paired with a second order finite difference scheme for non-uniform meshes to generate an error estimate for the second order finite difference scheme. The difference between the solutions obtained from the two schemes is used to generate an error estimate for the second order finite difference scheme. The schemes are tested in three cases for order of convergence and quality of error estimate. The first test case is boundary value ordinary differential equations, the second test case is 1D partial differential equations, and the third case is 2D partial differential equations. There were two methods to define boundary conditions in the 2D partial differential equation, either in Dirichlet or Neumann form. We found that both finite difference schemes had an experimental order of convergence consistent with the expected theoretical order of convergence in the boundary value ordinary differential equation and 1D partial differential equation (PDE) cases. This was also demonstrated in the 2D PDE case when using boundary conditions in Neumann form. Only the fourth order scheme showed experimentally and theoretically consistent convergence rates on 2D PDEs when using Dirichlet boundary conditions. The error estimate was within one order of magnitude of the true error for the second order finite difference scheme in all cases where the subintervals in the meshes were small enough to allow the Taylor's series, upon which the finite difference schemes are based, to hold. We conclude that the schemes can be used within a computational algorithm incorporating adaptive meshing and error control features.

Chapter 1

Introduction

Many scientific fields involve the solution of differential equations which may have no closed form solution. The applications can vary from modelling animal populations [1] to modelling brain tumor growth [2]. In those cases, special techniques called numerical methods must be used to get an approximate solution. These computational methods are able to obtain a solution whose error estimate is within within a specific user tolerance. Finite difference schemes are a type of method derived from Taylor's series. They approximate various derivatives of a function by using evaluations of that function at a set of finite discrete points called a mesh. The mesh may be spaced uniformly or non-uniformly. An advantage with non-uniform meshes is that they can generate a more accurate solution without adding extra points to a mesh. This is done by allowing clustering of mesh points in areas of high solution variance and allowing fewer points in areas of low solution variance. The points of the non-uniform mesh can be placed to evenly distribute the error of the approximation across the whole domain.

In order to generate an error estimate for a second order finite difference scheme, a solution of higher order is needed for comparison. Second order finite difference schemes for non-uniform meshes are common and the derivation of one is detailed in [3]. Higher order finite difference schemes for non-uniform meshes often come in the form of an algorithm [4]. That is, the finite difference scheme is not given in an explicit closed form; rather, given the appropriate parameters, software is used to compute the scheme. A major goal of this thesis is to derive an explicit representation for several fourth order finite difference schemes.

Good quality numerical software allows the user to specify a tolerance that the error estimate must satisfy. One way of generating an error estimate involves using two finite difference schemes with different orders of convergence. In this thesis, we derive a fourth order finite difference scheme for non-uniform meshes. This fourth order scheme is then coupled with a standard second order finite difference scheme for non-uniform meshes [3]. This allows us to to generate

an error estimate for the second order finite difference scheme. Both finite difference schemes are then tested for order of convergence and quality of error estimation on non-uniform meshes in three different settings. The first case considers boundary value ordinary differential equations (BVODEs). This was implemented in Scilab by applying both schemes separately to the test problems to yield systems of non-linear equations. The system was solved via the built-in Scilab function *fsolve* [5]. The second case considers 1D partial differential equations (PDEs) and was also implemented in Scilab. Both finite difference schemes were applied separately to the spatial variable of the 1D PDE to create a system of time dependent initial value ordinary differential equations (IVODEs). This was then solved via the scilab function *ode* [6]. The final test was an implementation in Fortran using BACOLI [7] that applied both finite difference schemes to the y variable of a 2D PDE, giving two systems of 1D PDEs. These were passed to BACOLI as a single system. Treating the 1D PDE system this way allows BACOLI to compute two solutions without running twice and guarantees BACOLI's adaptive refinement uses the same mesh points for both systems. We consider two different ways of treating the boundary conditions. The first approach is to define the boundary conditions in Neumann form [8]. The second approach is to define the conditions in Dirichlet form [8]. These forms will be explained later in the thesis.

The numerical experiments showed that both finite difference schemes yielded experimental rates of convergence consistent with their theoretical convergence rates in the BVODE and 1D PDE cases. The experimental rates of convergence for both finite difference schemes were also consistent with theoretical rates of convergence in the 2D PDE cases when using Neumann boundary conditions. The fourth order finite difference scheme also demonstrated a rate of convergence consistent with theory in the 2D PDE case when using Dirichlet boundary conditions. The second order finite difference scheme did not show an experimental rate of convergence consistent with the theoretical rate when using Dirichlet boundary conditions for the 2D PDE case. The error estimate was within an order of magnitude of the actual error of the second order finite difference scheme in all cases where the subintervals in the meshes were small enough to justify the use of the Taylor series upon which the finite difference schemes are based. We then conclude that the finite difference schemes can be used together to create numerical software capable of adaptive mesh refinement and error control.

The organization of this thesis is as follows. In Chapter 2 we give some background material relevant for subsequent sections of this thesis. In Chapter 3 the derivation of the fourth order finite difference scheme is detailed. In Chapter 4 details are provided on the test problems and their software implementations. In Chapter 5 the results of the numerical experiments are presented. In Chapter 6 extensions to this work are discussed. Appendix A produces a demonstration of how to use the software for the 2D PDE case.

Chapter 2

Background information

2.1 Ordinary and Partial Differential Equations

Many scientific fields use mathematical models and simulations to aid the investigative process. These models often involve systems of ordinary differential equations (ODEs), which describe how quantities change with respect to a single independent variable. To guarantee that a solution is unique, information is needed at one or more points within the domain. These are referred to as boundary conditions. If information is provided at only one point then the equation is an initial value ordinary differential equation (IVODE). If there is information for two points then the equation is a boundary value ordinary differential equation (BVODE).

An example of a general ODE is

$$u_t = u^2 u_{tt} + t,$$

where u is a function of t, and subscripts are used to represent derivatives. Note that these equations involve only t, u and its derivatives. Information on the boundary conditions will be discussed in section 2.3.

What separates ODEs from partial differential equations (PDEs) is the inclusion of additional independent variables. An example of a 1D PDE is

$$u_t = u_x + u_{xx} + u + x.$$

This is a 1D PDE since in addition to t, u has another independent variable, x. Information on the boundary conditions and the initial condition is detailed in section 2.3.

A 2D PDE will have two more independent variables in addition to t. An example of a 2D

PDE is

$$u_t = u_{xy} + u_x + u_y + x + y.$$

A 2D PDE has one more independent variable than a 1D PDE. The boundary conditions and initial condition are similar to that of a 1D PDE, and are explained further in section 2.3.

These equations can have a closed form solution which may or may not be easy to find, but often there is no exact solution and an approximate one must be obtained. This is often done through the use of computational software. Since the software computes an approximate solution there is the question of how accurate the approximation is. A good quality numerical software package often employs some form of error estimation and control. This error estimate can be used by an adaptive meshing method to determine a non-uniform mesh that will lead to a more efficient computation than would be obtained using fixed mesh methods. This is because the software can redistribute points in the domain to attempt to evenly distribute the error across the whole domain. The alternative is to add points across the whole domain, many of which may not be necessary. The error estimate can allow the software to produce a solution that should have an error that is consistent with a user provided tolerance.

2.2 Finite Difference Schemes

Finite difference schemes are a method for approximating a derivative of a function. Equations derived from Taylor's series are used to get an approximation of the derivative at a given point using function evaluations on a set of surrounding points. In our investigation we will apply finite difference schemes on a set of points that partitions a spatial domain into subintervals. This set of points is called a mesh. To generate the approximation of the derivative at a given point the function is evaluated at that point and a number of surrounding points. The sizes of the subintervals between the point and the surrounding points are used for a weighted average of the evaluations, giving the approximation of the derivative. For example, a centered finite difference scheme [4] does this by employing information at an equal number of points to the left and right of the point where the derivative is required.

The order of a finite difference scheme refers to how quickly the approximation will converge to the exact solution as the mesh subintervals become smaller. A finite difference scheme is said to be of order p if the error of the solution computed using the scheme behaves as $O(h^p)$, where h refers to the size of the mesh subintervals. In general, the order of a finite difference scheme will be one less than the number of points it employs in each step [4]. For example, a finite difference scheme that employs five points can be up to fourth order.

The point second from the left boundary point of the domain will only have one point to

the left of it. Similarly, the point one point in from the right boundary point will only have one one point to the right of it. Since higher order schemes will require more than one point to the left or right, this raises the issue that for these points there is not enough surrounding points to employ the centered finite difference scheme. These points close to the boundary points must be given special treatment using either forward or backward finite difference schemes [4]. Forward and backward finite difference schemes employ a different number of points to the left or right of the evaluation point. This allows the finite difference scheme to be adapted to work near the boundary points of the domain.

The set of mesh points that partitions the domain may be uniform, where the points are equally spaced, or non-uniform, where the spacing between the points can vary. The size of a mesh is determined by how many subintervals it has. So a mesh with six points would be of size five since there are five subintervals defined by the six points.

Numerical methods that only work on uniform meshes will work faster on uniform meshes than a method that can be used with non-uniform meshes. The trade-off is that the behavior of a given solution is often not uniform across the whole domain, so a uniform mesh may have many more points than necessary in some parts of the domain and fewer than necessary in other parts of the domain. This leads to large amounts of work where no appreciable accuracy is gained, greatly reducing the efficiency of the method. Thus the advantage of methods that can be used with non-uniform meshes is that the mesh refinement algorithm can cluster mesh points in regions of the domain where the solution changes rapidly and use fewer points overall. By using fewer points these methods will do less work in total and thus be faster than methods restricted to uniform meshes.

The finite difference schemes we consider in this thesis can be applied to one variable at a time. For example, we could consider a 2D PDE with three variables t, x, and y to then apply the mesh to the y domain to produce a system of 1D PDEs in terms of t and x. This system of 1D PDEs can then be passed to software capable of solving systems of 1D PDEs.

2.3 General Form of ODEs and PDEs

In this thesis we test our finite difference for convergence and quality of error estimate in three cases. The general form of each test case is discussed in this section.

2.3.1 The Boundary Value Ordinary Differential Equation Form

There are two main forms for the BVODEs in this thesis. Form one is

$$u_t(t) = f(t, u, u_{tt}),$$

on the domain $t \in [t_0, t_1]$ where t_0 is the left boundary point and t_1 is the right boundary point. Note t is the independent variable and u is a function of t. The boundary conditions are of the form

$$u(t_0) = a, \;\; u(t_1) = b,$$

where a and b are constants.

The second form we consider is

$$u_{tt}(t) = f(t, u),$$

on the domain $t \in [t_0, t_1]$. Again note that t is an independent variable and u is a function of t. The boundary conditions are of the form

$$u(t_0) = a, \;\; u(t_1) = b,$$

where a and b are constants.

In both cases the finite difference scheme is applied to the t domain. This produces a set of non-linear equations, with each equation associated with approximating the solution at a point t_i such that

$$t_0 < t_i < t_1.$$

2.3.2 The 1D Partial Differential Equation Form

There is one form for the 1D PDEs studied in this thesis. It is

$$u_t(t, x) = f(t, x, u, u_x, u_{xx}),$$

on the domain $t \in [t_0, t_1]$, $x \in [x_a, x_b]$. Here u is a function of t and x, t_0 is the initial value for t for which initial conditions are needed for u, t_1 is the final value of t, x_a is the left boundary point in the x domain, and x_b is the right boundary point in the x domain. The boundary conditions are of the form

$$u_t(t, x_a) = b_a(t), \;\; u_t(t, x_b) = b_b(t),$$

where b_a and b_b are functions of t. The boundary conditions, which involve the t derivatives of u, are chosen to be of this form so that they can be included in the system of ODEs (obtained after a finite difference scheme is applied to the x domain of the PDE) and passed to Scilabs *ode* function [6]. The finite difference scheme is applied to the x domain. This produces a set of IVODEs, with each IVODE approximating the solution along a line across the t domain. The initial conditions are of the form

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

where I is a function of x.

2.3.3 The 2D Partial Differential Equation Form

There is one form for the 2D PDEs in this thesis with two different ways of treating the boundary conditions associated with the y domain. The form of the 2D PDE is

$$u_t(t, x, y) = f(t, x, u, u_x, u_y, u_{xx}, u_{xy}, u_{yy}),$$

on the domain $t \in [t_0, t_1]$, $x \in [x_a, x_b]$, $y \in [y_a, y_b]$. Here u is a function of t, x, and y; t_0 is the initial value for t for which initial conditions are needed for u; t_1 is the final value of t; x_a is the left boundary point of the x domain; x_b is the right boundary point of the x domain; y_a is the left boundary point of the y domain; and y_b is the right boundary point of the y domain. The initial conditions are of the form

$$u(t_0, x, y) = I(x, y),$$

where I is a function of x and y. Boundary conditions for the x domain are assumed to be in the form that BACOLI requires [7], and are thus of the form

$$b_{x_a}(t, y, u, u_x) = 0, \;\; b_{x_b}(t, y, u, u_x) = 0.$$

We consider two different types of boundary conditions for the y domain. The first type are Neumann boundary conditions

$$u_t(t, x, y_a) = b_{y_a}(t, x), \;\; u_t(t, x, y_b) = b_{y_b}(t, x).$$

The second type of boundary conditions are similar to Dirichlet boundary conditions

$$u(t, x, y_a) = b_{1y_a}(t, x), \;\; u_x(t, x, y_a) = b_{2y_a}(t, x)$$

for y_a and

$$u(t, x, y_b) = b_{1y_b}(t, x), \;\; u_x(t, x, y_b) = b_{2y_b}(t, x)$$

for y_b. Note that if the PDE does not involve the cross derivitive term, u_{xy}, then the boundary conditions involving u_x can be omitted. There are two different forms for the boundary conditions considered in this case for the purposes of flexibility and comparison when approximating the cross derivative term u_{xy}.

The finite difference scheme is applied to the y domain, giving a set of 1D PDEs. Each PDE represents an approximation of the solution on a 2D surface in the (t, x) domain.

2.4 Software Used

BACOLI is a Fortran 77 software package used for the error controlled numerical solution of 1D PDE systems [7]. It employs spatial error estimation and control with adaptive mesh refinement [9]. This software will be used in solving 2D PDE systems. A finite difference scheme is applied to the y domain; the resultant system of 1D PDEs will then be solved by BACOLI. Detailed information on BACOLI can be found within [9].

Scilab is a software environment for mathematical computing [10] and will be used in solving ODE systems and 1D PDE systems. The scilab *fsolve* function [5] will be used when a finite difference scheme is applied to an ODE to obtain a system of non-linear equations . The *ode* function [6] will be used for the case of 1D PDEs. A finite difference scheme will be applied to the x domain and the resultant system of ODEs will then be solved by the Scilab function *ode*.

SymPy is a Python Library for computational algebra [11]. It will be used to solve the system of equations associated with deriving the fourth order finite difference scheme.

Chapter 3

Deriving a Fourth Order Variable Mesh Finite Difference Scheme

To generate an error estimate for a second order finite difference scheme, one approach is to employ a finite difference scheme of higher order. We choose a fourth order finite difference scheme since this will generate a good quality error estimate for the second order finite difference scheme, even on coarser meshes. We derive a fourth order centered finite difference scheme, two forward finite difference schemes, and two backwards finite difference schemes. In order to be of fourth order, each scheme will need to use function evaluations from five points [4]. Let those points be

$$\{a, b, c, d, e\},\ a < b < c < d < e.$$

Without loss of generality, expand about the center point, c. This gives a set of associated distances,

$$\{h_1, h_2, h_3, h_4\},\ h_1 = c - a,\ h_2 = c - b,\ h_3 = d - c,\ h_4 = e - c,$$

which can be used in Taylor series expansions of a given function f about c. The four expansions are

$$\begin{aligned}
f(a) &= f(c) - h_1 f'(c) + \frac{h_1^2 f''(c)}{2!} - \frac{h_1^3 f^{(3)}(c)}{3!} + \frac{h_1^4 f^{(4)}(c)}{4!} + O(h_1^5),\\
f(b) &= f(c) - h_2 f'(c) + \frac{h_2^2 f''(c)}{2!} - \frac{h_2^3 f^{(3)}(c)}{3!} + \frac{h_2^4 f^{(4)}(c)}{4!} + O(h_2^5),\\
f(d) &= f(c) + h_3 f'(c) + \frac{h_3^2 f''(c)}{2!} + \frac{h_3^3 f^{(3)}(c)}{3!} + \frac{h_3^4 f^{(4)}(c)}{4!} + O(h_3^5),\\
f(e) &= f(c) + h_4 f'(c) + \frac{h_4^2 f''(c)}{2!} + \frac{h_4^3 f^{(3)}(c)}{3!} + \frac{h_4^4 f^{(4)}(c)}{4!} + O(h_4^5).
\end{aligned} \tag{3.1}$$

In order to obtain a fourth order finite difference scheme that approximates a derivative of $f(c)$ we need

$$X = Af(a) + Bf(b) + Cf(c) + Df(d) + Ef(e) \tag{3.2}$$

where A, B ,C, D, and E are constants. Substituting the Taylor expansions from (3.1) into (3.2) gives

$$X = A[f(c) - h_1 f'(c) + \frac{h_1^2 f''(c)}{2!} - \frac{h_1^3 f^{(3)}(c)}{3!} + \frac{h_1^4 f^{(4)}(c)}{4!} + O(h_1^5)]$$

$$+B[f(c) - h_2 f'(c) + \frac{h_2^2 f''(c)}{2!} - \frac{h_2^3 f^{(3)}(c)}{3!} + \frac{h_2^4 f^{(4)}(c)}{4!} + O(h_2^5)]$$

$$+C[f(c)]$$

$$+D[f(c) + h_3 f'(c) + \frac{h_3^2 f''(c)}{2!} + \frac{h_3^3 f^{(3)}(c)}{3!} + \frac{h_3^4 f^{(4)}(c)}{4!} + O(h_3^5)]$$

$$+E[f(c) + h_4 f'(c) + \frac{h_4^2 f''(c)}{2!} + \frac{h_4^3 f^{(3)}(c)}{3!} + \frac{h_4^4 f^{(4)}(c)}{4!} + O(h_4^5)]$$

We next rearrange the right hand side of the above expression in terms of derivatives of $f(c)$ and think of X as five equations, each part dealing with a single derivative of $f(c)$. This yields the system

$$X_1 = (A + B + C + D + E)f(c),$$

$$X_2 = (-h_1 A - h_2 B + h_3 D + h_4 E)f'(c),$$

$$X_3 = (h_1^2 A + h_2^2 B + h_3^2 D + h_4^2 E)\frac{f''(c)}{2!},$$

$$X_4 = (-h_1^3 A - h_2^3 B + h_3^3 D + h_4^3 E)\frac{f^{(3)}(c)}{3!},$$

$$X_5 = (h_1^4 A + h_2^4 B + h_3^4 D + h_4^4 E)\frac{f^{(4)}(c)}{4!},$$

with associated error

$$O(h_1^5) + O(h_2^5) + O(h_3^5) + O(h_4^5).$$

We want (3.2) to give an approximation to $f'(c)$, therefore we must have $X_2 = 1$ with all other $X_i = 0$. To do this we need to find A, B ,C, D, and E in terms of h_1, h_2, h_3, and h_4 that satisfy the necessary condition. The system to be solved to obtain the fourth order approximation for

$f'(c)$ is

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -h_1 & -h_2 & 0 & h_3 & h_4 \\ h_1^2 & h_2^2 & 0 & h_3^2 & h_4^2 \\ -h_1^3 & -h_2^3 & 0 & h_3^3 & h_4^3 \\ h_1^4 & h_2^4 & 0 & h_3^4 & h_4^4 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

To obtain a fourth order finite difference scheme approximating the second derivative of $f(c)$, (3.2) needs to be satisfied when $X = f''(c)$. To have (3.2) give an approximation to $f''(c)$ we must have $X_3 = 2$ when all other $X_i = 0$. The system to be solved for the approximation of $f''(c)$ is thus

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -h_1 & -h_2 & 0 & h_3 & h_4 \\ h_1^2 & h_2^2 & 0 & h_3^2 & h_4^2 \\ -h_1^3 & -h_2^3 & 0 & h_3^3 & h_4^3 \\ h_1^4 & h_2^4 & 0 & h_3^4 & h_4^4 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{bmatrix}.$$

Solutions to these systems are listed below. The associated systems and their solutions for expansions about the other points are also listed. Solving these systems was done using the python library *SymPy* [11]. We explain how the above approach is modified to obtain finite difference approximations at the other points a, b, d, and e in the remaining sections of this chapter.

3.1 Fourth Order Finite Difference Approximation of $f'(c)$

The coefficients for the approximation of $f'(c)$ are

$$A = \frac{h_2h_3h_4}{h_1(h_1^3 - h_1^2h_2 + h_1^2h_3 + h_1^2h_4 - h_1h_2h_3 - h_1h_2h_4 + h_1h_3h_4 - h_2h_3h_4)},$$

$$B = \frac{-h_1h_3h_4}{h_2(h_1h_2^2 + h_1h_2h_3 + h_1h_2h_4 + h_1h_3h_4 - h_2^3 - h_2^2h_3 - h_2^2h_4 - h_2h_3h_4)},$$

$$C = \frac{1}{h_1} + \frac{1}{h_2} - \frac{1}{h_3} - \frac{1}{h_4},$$

$$D = \frac{-h_1h_2h_4}{h_3(h_1h_2h_3 - h_1h_2h_4 + h_1h_3^2 - h_1h_3h_4 + h_2h_3^2 - h_2h_3h_4 + h_3^3 - h_3^2h_4)},$$

$$E = \frac{h_1h_2h_3}{h_4(h_1h_2h_3 - h_1h_2h_4 + h_1h_3h_4 - h_1h_4^2 + h_2h_3h_4 - h_2h_4^2 + h_3h_4^2 - h_4^3)}.$$

They are used as follows,

$$f'(c) = Af(a) + Bf(b) + Cf(c) + Df(d) + Ef(e).$$

3.2 Fourth Order Finite Difference Approximation of $f''(c)$

The coefficients for the approximation of $f''(c)$ are

$$A = \frac{2(-h_2h_3 - h_2h_4 + h_3h_4)}{h_1(h_1^3 - h_1^2h_2 + h_1^2h_3 + h_1^2h_4 - h_1h_2h_3 - h_1h_2h_4 + h_1h_3h_4 - h_2h_3h_4)}$$

$$B = \frac{2(h_1h_3 + h_1h_4 - h_3h_4)}{h_2(h_1h_2^2 + h_1h_2h_3 + h_1h_2h_4 + h_1h_3h_4 - h_2^3 - h_2^2h_3 - h_2^2h_4 - h_2h_3h_4)}$$

$$C = \frac{2}{h_3h_4} - \frac{2}{h_2h_4} - \frac{2}{h_2h_3} - \frac{2}{h_1h_4} - \frac{2}{h_1h_3} + \frac{2}{h_1h_2}$$

$$D = \frac{2(h_1h_2 - h_1h_4 - h_2h_4)}{h_3(h_1h_2h_3 - h_1h_2h_4 + h_1h_3^2 - h_1h_3h_4 + h_2h_3^2 - h_2h_3h_4 + h_3^3 - h_3^2h_4)}$$

$$E = \frac{2(-h_1h_2 + h_1h_3 + h_2h_3)}{h_4(h_1h_2h_3 - h_1h_2h_4 + h_1h_3h_4 - h_1h_4^2 + h_2h_3h_4 - h_2h_4^2 + h_3h_4^2 - h_4^3)}$$

They are used as follows,

$$f''(c) = Af(a) + Bf(b) + Cf(c) + Df(d) + Ef(e).$$

3.3 Fourth Order Finite Difference Approximation of $f'(a)$

Finding the fourth order finite difference scheme centered about a is done in a similar manner as it was for c. The Taylor expansions about a are found for the other four points. These expansions are then substituted into (3.2) with $X = f'(a)$. The system is then rearranged in the same way as it was for c, yielding the system

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & h_1 & h_2 & h_3 & h_4 \\ 0 & h_1^2 & h_2^2 & h_3^2 & h_4^2 \\ 0 & h_1^3 & h_2^3 & h_3^3 & h_4^3 \\ 0 & h_1^4 & h_2^4 & h_3^4 & h_4^4 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

The solution is

$$A = -\frac{1}{h_1} - \frac{1}{h_2} - \frac{1}{h_3} - \frac{1}{h_4},$$

$$B = \frac{-h_2h_3h_4}{h_1(h_1^3 - h_1^2h_2 - h_1^2h_3 - h_1^2h_4 + h_1h_2h_3 + h_1h_2h_4 + h_1h_3h_4 - h_2h_3h_4)},$$

$$C = \frac{h_1h_3h_4}{h_2(h_1h_2^2 - h_1h_2h_3 - h_1h_2h_4 + h_1h_3h_4 - h_2^3 + h_2^2h_3 + h_2^2h_4 - h_2h_3h_4)},$$

$$D = \frac{-h_1h_2h_4}{h_3(h_1h_2h_3 - h_1h_2h_4 - h_1h_3^2 + h_1h_3h_4 - h_2h_3^2 + h_2h_3h_4 + h_3^3 - h_3^2h_4)},$$

$$E = \frac{h_1h_2h_3}{h_4(h_1h_2h_3 - h_1h_2h_4 - h_1h_3h_4 + h_1h_4^2 - h_2h_3h_4 + h_2h_4^2 + h_3h_4^2 - h_4^3)}.$$

They are used as follows,

$$f'(a) = Af(a) + Bf(b) + Cf(c) + Df(d) + Ef(e).$$

3.4 Fourth Order Finite Difference Approximation of $f''(a)$

Finding the fourth order finite difference scheme centered about a is done in a similar manner as it was for c. The Taylor expansions about a are found for the other four points. These expansions are then substituted into (3.2) with $X = f''(a)$. The system is then rearranged in the same way as it was for c, yielding the system

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & h_1 & h_2 & h_3 & h_4 \\ 0 & h_1^2 & h_2^2 & h_3^2 & h_4^2 \\ 0 & h_1^3 & h_2^3 & h_3^3 & h_4^3 \\ 0 & h_1^4 & h_2^4 & h_3^4 & h_4^4 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{bmatrix}.$$

The solution is

$$A = \frac{2}{h_3h_4} + \frac{2}{h_2h_4} + \frac{2}{h_2h_3} + \frac{2}{h_1h_4} + \frac{2}{h_1h_3} + \frac{2}{h_1h_2},$$

$$B = \frac{2(h_2h_3 + h_2h_4 + h_3h_4)}{h_1(h_1^3 - h_1^2h_2 - h_1^2h_3 - h_1^2h_4 + h_1h_2h_3 + h_1h_2h_4 + h_1h_3h_4 - h_2h_3h_4)},$$

$$C = \frac{-2(h_1h_3 + h_1h_4 + h_3h_4)}{h_2(h_1h_2^2 - h_1h_2h_3 - h_1h_2h_4 + h_1h_3h_4 - h_2^3 + h_2^2h_3 + h_2^2h_4 - h_2h_3h_4)},$$

$$D = \frac{2(h_1h_2 + h_1h_4 + h_2h_4)}{h_3(h_1h_2h_3 - h_1h_2h_4 - h_1h_3^2 + h_1h_3h_4 - h_2h_3^2 + h_2h_3h_4 + h_3^3 - h_3^2h_4)},$$

$$E = \frac{-2(h_1h_2 + h_1h_3 + h_2h_3)}{h_4(h_1h_2h_3 - h_1h_2h_4 - h_1h_3h_4 + h_1h_4^2 - h_2h_3h_4 + h_2h_4^2 + h_3h_4^2 - h_4^3)}.$$

They are used as follows,

$$f''(a) = Af(a) + Bf(b) + Cf(c) + Df(d) + Ef(e).$$

3.5 Fourth Order Finite Difference Approximation of $f'(b)$

Finding the fourth order finite difference scheme centered about b is done in a similar manner as it was for c. The Taylor expansions about b are found for the other four points. These expansions are then substituted into (3.2) with $X = f'(b)$. The system is then rearranged in the same way as it was for c, yielding the system

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -h_1 & 0 & h_2 & h_3 & h_4 \\ h_1^2 & 0 & h_2^2 & h_3^2 & h_4^2 \\ -h_1^3 & 0 & h_2^3 & h_3^3 & h_4^3 \\ h_1^4 & 0 & h_2^4 & h_3^4 & h_4^4 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

The solution is

$$A = \frac{-h_2h_3h_4}{h_1(h_1^3 + h_1^2h_2 + h_1^2h_3 + h1^2h_4 + h_1h_2h_3 + h_1h_2h_4 + h_1h_3h_4 + h_2h_3h_4)},$$

$$B = \frac{1}{h_1} - \frac{1}{h_2} - \frac{1}{h_3} - \frac{1}{h_4},$$

$$C = \frac{h_1h_3h_4}{h_2(h_1h_2^2 - h_1h_2h_3 - h_1h_2h_4 + h_1h_3h_4 + h_2^3 - h_2^2h_3 - h_2^2h_4 + h_2h_3h_4)},$$

$$D = \frac{-h_1h_2h_4}{h_3(h_1h_2h_3 - h_1h_2h_4 - h_1h_3^2 + h_1h_3h_4 + h_2h_3^2 - h_2h_3h_4 - h_3^3 + h_3^2h_4)},$$

$$E = \frac{h_1h_2h_3}{h_4(h_1h_2h_3 - h_1h_2h_4 - h_1h_3h_4 + h_1h_4^2 + h_2h_3h_4 - h_2h_4^2 - h_3h_4^2 + h_4^3)}.$$

They are used as follows,

$$f'(b) = Af(a) + Bf(b) + Cf(c) + Df(d) + Ef(e).$$

3.6 Fourth Order Finite Difference Approximation of $f''(b)$

Finding the fourth order finite difference scheme centered about b is done in a similar manner as it was for c. The Taylor expansions about b are found for the other four points. These expansions are then substituted into (3.2) with $X = f''(b)$. The system is then rearranged in the same way

as it was for c, yielding the system

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -h_1 & 0 & h_2 & h_3 & h_4 \\ h_1^2 & 0 & h_2^2 & h_3^2 & h_4^2 \\ -h_1^3 & 0 & h_2^3 & h_3^3 & h_4^3 \\ h_1^4 & 0 & h_2^4 & h_3^4 & h_4^4 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{bmatrix}.$$

The solution is

$$A = \frac{2(h_2h_3 + h_2h_4 + h_3h_4)}{h_1(h_1^3 + h_1^2h_2 + h_1^2h_3 + h_1^2h_4 + h_1h_2h_3 + h_1h_2h_4 + h_1h_3h_4 + h_2h_3h_4)},$$

$$B = \frac{2}{h_3h_4} + \frac{2}{h_2h_4} + \frac{2}{h_2h_3} - \frac{2}{h_1h_4} - \frac{2}{h_1h_3} - \frac{2}{h_1h_2},$$

$$C = \frac{2(-h_1h_3 - h_1h_4 + h_3h_4)}{h_2(h_1h_2^2 - h_1h_2h_3 - h_1h_2h_4 + h_1h_3h_4 + h_2^3 - h_2^2h_3 - h_2^2h_4 + h_2h_3h_4)},$$

$$D = \frac{2(h_1h_2 + h_1h_4 - h_2h_4)}{h_3(h_1h_2h_3 - h_1h_2h_4 - h_1h_3^2 + h_1h_3h_4 + h_2h_3^2 - h_2h_3h_4 - h_3^3 + h_3^2h_4)},$$

$$E = \frac{2(-h_1h_2 - h_1h_3 + h_2h_3)}{h_4(h_1h_2h_3 - h_1h_2h_4 - h_1h_3h_4 + h_1h_4^2 + h_2h_3h_4 - h_2h_4^2 - h_3h_4^2 + h_4^3)}.$$

They are used as follows,

$$f''(b) = Af(a) + Bf(b) + Cf(c) + Df(d) + Ef(e).$$

3.7 Fourth Order Finite Difference Approximation of $f'(d)$

Finding the fourth order finite difference scheme centered about d is done in a similar manner as it was for c. The Taylor expansions about d are found for the other four points. These expansions are then substituted into (3.2) with $X = f'(d)$. The system is then rearranged in the same way as it was for c, yielding the system

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -h_1 & -h_2 & -h_3 & 0 & h_4 \\ h_1^2 & h_2^2 & h_3^2 & 0 & h_4^2 \\ -h_1^3 & -h_2^3 & -h_3^3 & 0 & h_4^3 \\ h_1^4 & h_2^4 & h_3^4 & 0 & h_4^4 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

The solution is

$$A = \frac{-h_2h_3h_4}{h_1(h_1^3 - h_1^2h_2 - h_1^2h_3 + h_1^2h_4 + h_1h_2h_3 - h_1h_2h_4 - h_1h_3h_4 + h_2h_3h_4)},$$

$$B = \frac{h_1h_3h_4}{h_2(h_1h_2^2 - h_1h_2h_3 + h_1h_2h_4 - h_1h_3h_4 - h_2^3 + h_2^2h_3 - h_2^2h_4 + h_2h_3h_4)},$$

$$C = \frac{-h_1h_2h_4}{h_3(h_1h_2h_3 + h_1h_2h_4 - h_1h_3^2 - h_1h_3h_4 - h_2h_3^2 - h_2h_3h_4 + h_3^3 + h_3^2h_4)},$$

$$D = \frac{1}{h_1} + \frac{1}{h_2} + \frac{1}{h_3} - \frac{1}{h_4},$$

$$E = \frac{h_1h_2h_3}{h_4(h_1h_2h_3 + h_1h_2h_4 + h_1h_3h_4 + h_1h_4^2 + h_2h_3h_4 + h_2h_4^2 + h_3h_4^2 + h_4^3)}.$$

They are used as follows,

$$f'(d) = Af(a) + Bf(b) + Cf(c) + Df(d) + Ef(e).$$

3.8 Fourth Order Finite Difference Approximation of $f''(d)$

Finding the fourth order finite difference scheme centered about d is done in a similar manner as it was for c. The Taylor expansions about d are found for the other four points. These expansions are then substituted into (3.2) with $X = f''(d)$. The system is then rearranged in same way as it was for c, yielding the system

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -h_1 & -h_2 & -h_3 & 0 & h_4 \\ h_1^2 & h_2^2 & h_3^2 & 0 & h_4^2 \\ -h_1^3 & -h_2^3 & -h_3^3 & 0 & h_4^3 \\ h_1^4 & h_2^4 & h_3^4 & 0 & h_4^4 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{bmatrix}.$$

The solution is

$$A = \frac{2(h_2h_3 - h_2h_4 - h_3h_4)}{h_1(h_1^3 - h_1^2h_2 - h_1^2h_3 + h_1^2h_4 + h_1h_2h_3 - h_1h_2h_4 - h_1h_3h_4 + h_2h_3h_4)},$$

$$B = \frac{2(-h_1h_3 + h_1h_4 + h_3h_4)}{h_2(h_1h_2^2 - h_1h_2h_3 + h_1h_2h_4 - h_1h_3h_4 - h_2^3 + h_2^2h_3 - h_2^2h_4 + h_2h_3h_4)},$$

$$C = \frac{2(h_1h_2 - h_1h_4 - h_2h_4)}{h_3(h_1h_2h_3 + h_1h_2h_4 - h_1h_3^2 - h_1h_3h_4 - h_2h_3^2 - h_2h_3h_4 + h_3^3 + h_3^2h_4)},$$

$$D = -\frac{2}{h_3h_4} - \frac{2}{h_2h_4} + \frac{2}{h_2h_3} - \frac{2}{h_1h_4} + \frac{2}{h_1h_3} + \frac{2}{h_1h_2},$$

$$E = \frac{2(h_1h_2 + h_1h_3 + h_2h_3)}{h_4(h_1h_2h_3 + h_1h_2h_4 + h_1h_3h_4 + h_1h_4^2 + h_2h_3h_4 + h_2h_4^2 + h_3h_4^2 + h_4^3)}.$$

They are used as follows,

$$f''(d) = Af(a) + Bf(b) + Cf(c) + Df(d) + Ef(e).$$

3.9 Fourth Order Finite Difference Approximation of $f'(e)$

Finding the fourth order finite difference scheme centered about e is done in a similar manner as it was for c. The Taylor expansions about e are found for the other four points. These expansions are then substituted into (3.2) with $X = f'(e)$. The system is then rearranged in the same way as it was for c, yielding the system

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -h_1 & -h_2 & -h_3 & -h_4 & 0 \\ h_1^2 & h_2^2 & h_3^2 & h_4^2 & 0 \\ -h_1^3 & -h_2^3 & -h_3^3 & -h_4^3 & 0 \\ h_1^4 & h_2^4 & h_3^4 & h_4^4 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

The solution is

$$A = \frac{h_2h_3h_4}{h_1(h_1^3 - h_1^2h_2 - h_1^2h_3 - h_1^2h_4 + h_1h_2h_3 + h_1h_2h_4 + h_1h_3h_4 - h_2h_3h_4)},$$

$$B = \frac{-h_1h_3h_4}{h_2(h_1h_2^2 - h_1h_2h_3 - h_1h_2h_4 + h_1h_3h_4 - h_2^3 + h_2^2h_3 + h_2^2h_4 - h_2h_3h_4)},$$

$$C = \frac{h_1h_2h_4}{h_3(h_1h_2h_3 - h_1h_2h_4 - h_1h_3^2 + h_1h_3h_4 - h_2h_3^2 + h_2h_3h_4 + h_3^3 - h_3^2h_4)},$$

$$D = \frac{-h_1h_2h_3}{h_4(h_1h_2h_3 - h_1h_2h_4 - h_1h_3h_4 + h_1h_4^2 - h_2h_3h_4 + h_2h_4^2 + h_3h_4^2 - h_4^3)},$$

$$E = \frac{1}{h_1} + \frac{1}{h_2} + \frac{1}{h_3} + \frac{1}{h_4}.$$

They are used as follows,

$$f'(e) = Af(a) + Bf(b) + Cf(c) + Df(d) + Ef(e).$$

3.10 Fourth Order Finite Difference Approximation of $f''(e)$

Finding the fourth order finite difference scheme centered about e is done in a similar manner as it was for c. The Taylor expansions about e are found for the other four points. These expansions are then substituted into (3.2) with $X = f''(e)$. The system is then rearranged in the same way as it was for c, yielding the system

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -h_1 & -h_2 & -h_3 & -h_4 & 0 \\ h_1^2 & h_2^2 & h_3^2 & h_4^2 & 0 \\ -h_1^3 & -h_2^3 & -h_3^3 & -h_4^3 & 0 \\ h_1^4 & h_2^4 & h_3^4 & h_4^4 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{bmatrix}.$$

The solution is

$$A = \frac{2(h_2h_3 + h_2h_4 + h_3h_4)}{h_1(h_1^3 - h_1^2h_2 - h_1^2h_3 - h_1^2h_4 + h_1h_2h_3 + h_1h_2h_4 + h_1h_3h_4 - h_2h_3h_4)},$$

$$B = \frac{-2(h_1h_3 + h_1h_4 + h_3h_4)}{h_2(h_1h_2^2 - h_1h_2h_3 - h_1h_2h_4 + h_1h_3h_4 - h_2^3 + h_2^2h_3 + h_2^2h_4 - h_2h_3h_4)},$$

$$C = \frac{2(h_1h_2 + h_1h_4 + h_2h_4)}{h_3(h_1h_2h_3 - h_1h_2h_4 - h_1h_3^2 + h_1h_3h_4 - h_2h_3^2 + h_2h_3h_4 + h_3^3 - h_3^2h_4)},$$

$$D = \frac{-2(h_1h_2 + h_1h_3 + h_2h_3)}{h_4(h_1h_2h_3 - h_1h_2h_4 - h_1h_3h_4 + h_1h_4^2 - h_2h_3h_4 + h_2h_4^2 + h_3h_4^2 - h_4^3)},$$

$$E = \frac{2}{h_3h_4} + \frac{2}{h_2h_4} + \frac{2}{h_2h_3} + \frac{2}{h_1h_4} + \frac{2}{h_1h_3} + \frac{2}{h_1h_2}.$$

They are used as follows,

$$f''(e) = Af(a) + Bf(b) + Cf(c) + Df(d) + Ef(e).$$

Chapter 4

Test Problems and Software

In this Chapter we discuss the software development that is needed in order to conduct the numerical experiments that will verify order of convergence for both of our finite difference schemes and the quality of the error estimate for the second order finite difference scheme. The first and second sets of experiments are implementations in Scilab which apply the finite difference schemes to ODEs and 1D PDEs. The third set of experiments apply the finite difference schemes to 2D PDEs yielding systems of 1D PDEs that are solved using the software package BACOLI. Examples of how to us the source code are provided in Appendix A.

4.1 BVODE Case

To verify the experimental rate of convergence of both finite difference schemes and the quality of the error estimate for the second order finite difference scheme a Scilab implementation was developed for two problems.

The first problem is

$$u_{tt} = \frac{3}{2}u^2, \tag{4.1}$$

with boundary conditions

$$u(0) = 4, \;\; u(1) = 1.$$

The domain is

$$t \in (0, 1),$$

and exact solution

$$u = \frac{4}{(1+t)^2}.$$

The second problem is

$$u_t = 6u^2 t \tag{4.2}$$

with boundary conditions

$$u(0) = \frac{1}{28},\ u(2.5) = \frac{1}{28 - 3(2.5)^2}.$$

The domain is

$$t \in (0, 2.5),$$

and exact solution

$$u = \frac{1}{28 - 3t^2}.$$

This implementation sets up two separate functions to apply the finite difference schemes to the t domain. These functions serve to discretize the ODE into a system of equations; the solution of each equation gives an approximation to the solution of the ODE at a single point in the t domain. The function for applying the fourth order scheme is separated into five different major sections. Two sections are dedicated to implementing the equations that correspond to the boundary conditions, two sections are dedicated to implementing the equations associated with applying the forward and backward finite difference schemes for the points just inside the boundaries, and the remaining section contains a loop for implementing the equations that correspond to applying the finite difference schemes at all of the central points. The function for the second order scheme is similar, but has only three major sections, two sections for the boundary conditions and one section containing a loop for the central points.

The same mesh is passed to both functions so that an the second and fourth order derivative approximations will be obtained at the same points in the t domain, and thus an error estimate can be generated for the second order solution. The solution to these systems is obtained by using the Scilab *fsolve* function with a default tolerance of 10^{-10}, which solves the systems using the Powell hybrid method [5]. This is an iterative algorithm for finding local minimums of a function. The error estimate is calculated by subtracting the solution associated with the fourth order scheme from the solution associated with the second order scheme. The implementation is independent of any specific mesh or domain, and supports both uniform and non-uniform meshes.

4.2 1D PDE Case

The 1D experiments will be conducted to verify that both finite difference schemes maintain their theoretical order of convergence and quality of error estimate in more difficult cases. The

first test PDE is

$$u_t = \frac{1}{4}(\epsilon u_{xx} - u u_x), \tag{4.3}$$

which is the 1D Burgers equation. An ϵ value of 0.01 was used and the domain is

$$x \in (0,1), \;\; t \in (0,1).$$

The boundary conditions

$$u_t(t,0) = \frac{\sec((\frac{1}{4\epsilon})(\frac{-t}{8} - \frac{1}{4}))^2}{64\epsilon}, \;\; u_t(t,1) = \frac{\sec((\frac{1}{4\epsilon})(\frac{-t}{8} + \frac{3}{4}))^2}{64\epsilon},$$

were used. Initial conditions were

$$u(0,x) = \frac{1}{2} - \frac{1}{2}\tanh\Big((\frac{1}{4\epsilon})(x - \frac{1}{4})\Big).$$

The exact solution is [12]

$$u = \frac{1}{2} - \frac{1}{2}\tanh\Big((\frac{1}{4\epsilon})(x - \frac{t}{8} - \frac{1}{4})\Big).$$

The second test PDE is

$$u_t = u_{xx} + \pi^2 \sin(\pi x), \tag{4.4}$$

The domain is

$$x \in (0,1), \;\; t \in (0,1).$$

The boundary conditions are

$$u_t(t,0) = 0, \;\; u_t(t,1) = 0.$$

The initial conditions are

$$u(0,x) = 1.$$

The exact solution is

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

The implementation employed for solving 1D PDEs is similar to the BVODE implementation. There are two separate functions associated with applying the finite difference schemes to the PDE with the same structure as the BVODE case discussed in the previous section. These functions discretize the x domain of the PDE, giving a system of IVODEs in which each IVODE represents a line across the x domain. This approach is known as the method of lines. The system

of IVODEs is solved by the Scilab IVODE solver, called *ode*, which uses the lsoda package by default [6]. A tolerance of 10^{-8} was used. The error estimate is calculated as the difference between the two solutions. The implementation is independent of any specific mesh or domain, and supports both uniform and non-uniform meshes.

4.3 2D PDE Case

The purpose of the 2D experiments is to verify that both finite difference schemes maintain their order of convergence in the 2D case. The experiments will also attempt to verify the quality of the error estimate in 2D.

The fourth and second order finite difference schemes were used along with the software package BACOLI to create FORTRAN 77 software that solves problems of the form described in Chapter 2. The finite difference schemes are applied to the y domain in these problems. The fourth and second order schemes are used together in this case. They are applied within a single function to generate two distinct systems of 1D PDEs which are passed to BACOLI as a single system. BACOLI's adaptive meshing algorithm is thus forced to employ the same mesh points for both systems. The error estimate is formed by subtracting the solution associated with the fourth order scheme from the solution associated with the second order scheme.

The first test problem is

$$u_t = \epsilon(u_{xx} + u_{yy}) - u(u_x + u_y), \tag{4.5}$$

which is the 2D Burgers equation [12]. We will consider $\epsilon = 0.1$. The domain for this problem is

$$x \in (0,1), \;\; y \in (0,1), \;\; t \in (0,1).$$

The initial conditions are

$$u(0,x,y) = \frac{1}{1 + e^{\frac{x+y}{2\epsilon}}}.$$

The boundary condition for $x = 0$ is

$$u(t,0,y) - \frac{1}{1 + e^{\frac{y-t}{2\epsilon}}} = 0.$$

The boundary condition for $x = 1$ is

$$u(t,1,y) - \frac{1}{1 + e^{\frac{1+y-t}{2\epsilon}}} = 0.$$

The Neumann boundary condition for $y = 0$ is

$$u_t(t, x, 0) = \frac{e^{\frac{x-t}{2\epsilon}}}{2\epsilon(1 + e^{\frac{x-t}{2\epsilon}})^2},$$

and for $y = 1$

$$u_t(t, x, 1) = \frac{e^{\frac{1+x-t}{2\epsilon}}}{2\epsilon(1 + e^{\frac{1+x-t}{2\epsilon}})^2}.$$

Since (4.5) does not contain the cross derivative, u_{xy}, the second method for defining boundary conditions becomes entirely Dirichlet. For $y = 0$ this gives

$$u(t, x, 0) = \frac{1}{1 + e^{\frac{x-t}{2\epsilon}}}.$$

For $y = 1$ the boundary condition is

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

The exact solution is

$$u = \frac{1}{1 + e^{\frac{x+y-t}{2\epsilon}}}.$$

The second test problem is problem 4 from [12], which is

$$u_t = (L_1 + L_2 + L_3)u + f(x, y, t), \tag{4.6}$$

where

$$L_1 = (x^2 + 1)u_{xx} + x, \;\; L_2 = (y^2 + 1)u_{yy} + yu_y + y, \;\; L_3 = u_{xy},$$

and f is chosen so that the exact solution is

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

The domain of this problem is

$$x \in (0, 1), \;\; y \in (0, 1), \;\; t \in (0, 1).$$

The initial conditions are

$$u(0, x, y) = 2\sin(\pi x)\sin(\pi y).$$

Boundary conditions for the x domain are

$$u(t,0,y) = 0, \;\; u(t,1,y) = 0,$$

The Neumann boundary conditions at $y = 0$ and $y = 1$ are

$$u_t(t,x,0) = 0, \;\; u_t(t,x,1) = 0.$$

Since (4.6) does contain the cross derivative, u_{xy}, the second type of boundary conditions is not entirely Dirichlet. In addition to the Dirichlet conditions, it requires boundary conditions that specify u_x. For $x = 0$ the boundary conditions are

$$u(t,x,0) = 0, \;\; u_x(t,x,0) = 0.$$

For $x = 1$ the boundary conditions are

$$u(t,x,1) = 0, \;\; u_x(t,x,1) = 0.$$

The mesh in the x dimension is determined adaptively by BACOLI such that the estimated error is equally distributed across the domain and such that a sufficient number of mesh points are chosen so that the user tolerance is satisfied. BACOLI is called with an error tolerance of 10^{-8} so that the error from BACOLI does not effect the error of the finite difference schemes that is used to discretize the y domain. The mesh in the y dimension must be provided and remains fixed; this mesh may be non-uniform.

Chapter 5

Results

In this Chapter we discuss the results of our three different sets of experiments. Recall that the point of these experiments is to allow us to experimentally assess the order of convergence of the finite difference schemes as well as the quality of the error estimate for the second order scheme. Recall also that the error estimate is calculated as the difference between the second and fourth order solutions at each mesh point. The first set of tests are on the simple case of applying both finite difference schemes to BVODEs. The second set of experiments are for 1D PDEs, and the third set of experiments are for 2D PDEs. These experiments together show that the finite difference schemes generally exhibit an experimental order of convergence consistent with their expected theoretical order in simple cases and maintain this into more difficult cases. The quality of the error estimate is also maintained in more difficult cases.

5.1 Boundary Value Ordinary Differential Equation Results

The application of the finite difference schemes to a BVODE leads to a system of non-linear equations which is solved by the Scilab function *fsolve* [5]. Let a and b be the bounds of the domain being discretized. Then the mesh of $N = m + 1$ subintervals is defined by $x_1 = a$, $x_{m+1} = b$, $x_{\frac{m}{2}+1} = \frac{b-a}{2} + a$, with points to the left of the midpoint defined as $x_{\frac{m}{2}-i+3} = \frac{b-a}{i} + a$ and points to the right of the midpoint defined as $x_{\frac{m}{2}+i-1} = b - \frac{b-a}{i}$ for $i = 3 : \frac{m}{2} + 1$. This produces a symmetric but non-uniform mesh with points clustered closer to the boundary points. An example of this can be seen in Figure 5.1. To obtain a finer mesh, points are added by placing a new mesh point in between two existing mesh points, i.e. each subinterval of the original mesh is halved. Thus, when going from a mesh of size $N = 20$ to a mesh of size $N = 40$, the size of all subintervals is cut in half. The shape of the mesh was chosen arbitrarily, and the same mesh

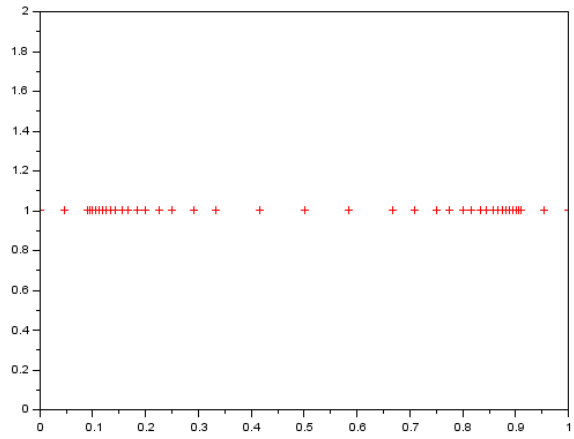

Figure 5.1: An example mesh with 40 subintervals. Each red cross represents a mesh point. The points cluster closer to the boundary conditions than the center of the domain.

is used in all BVODE test cases.

For each mesh point the exact error is computed as the absolute value of the difference between the true solution and the approximate solution. The average error is calculated as the mean of these values, and the maximum error is determined as the largest absolute error. Error ratios can then be calculated to experimentally determine the convergence rate of a given scheme. The average error ratio is calculated by dividing the average error of the solution of the previous mesh by the average error of the solution on the current mesh. A similar calculation is performed to compute the maximum error ratio. The rate of convergence is $-\log_2$ of these ratios. This is because the error behaves as $O(h^p)$; thus when the subintervals are halved the new error becomes $(\frac{h}{2})^p$, so the error is reduced to $(\frac{1}{2})^p$ of the previous error.

Solutions were computed for meshes of size N=20, 40, 80, 160, 320, and 640. The error ratios were calculated as detailed above, with the pairs of meshes being 20/40, 40/80, 80/160, 160/320, and 320/640. Table 5.1 provides error ratios for both finite difference schemes used to solve (4.1). Table 5.1 shows that the second order finite difference scheme has average and maximum error ratios converge to $\frac{1}{4} = \frac{1}{2^2}$. This indicates that the second order finite difference scheme is exhibiting second order convergence. The fourth order finite difference scheme has average and maximum error ratios of about $\frac{1}{16} = \frac{1}{2^4}$ which indicates it is exhibiting fourth order convergence.

Table 5.2 presents average errors and shows the fourth order scheme was significantly more accurate than the second order scheme. This leads to a good quality error estimate, with all estimates being within one order of magnitude of the true error for the second order finite difference scheme. Table 5.3 demonstrates this behavior for the maximum errors; the estimates are all in agreement to at least one significant digit with the true error. A graph of the exact solution for (4.1) can be seen in Figure 5.2.

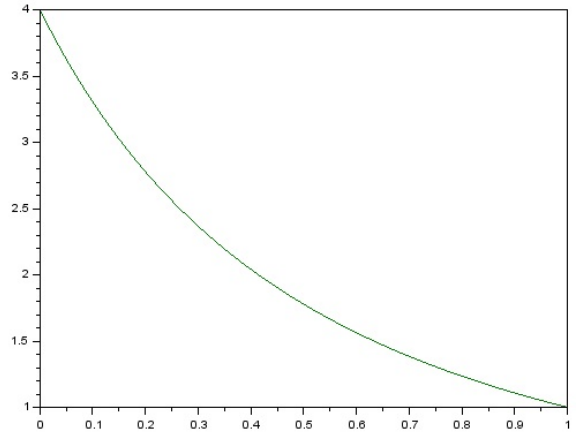

Figure 5.2: Graph of exact solution for (4.1).

Mesh Pairs	2nd Avg Ratio	2nd Max Ratio	4th Avg Ratio	4th Max Ratio
20/40	0.2596	0.2524	0.0392	0.0644
40/80	0.2539	0.2506	0.0172	0.0433
80/160	0.2518	0.2501	0.1097	0.0366
160/320	0.2508	0.2500	0.0999	0.0698
320/640	0.2504	0.2500	0.0697	0.0684

Table 5.1: Convergence ratios for (4.1). The ratios are obtained by dividing the error of the numerical solution computed on the previous mesh by the error of the numerical solution computed for the current mesh. For example, the data for 20/40 was obtained by dividing the error for the mesh with 20 subintervals by the error for the mesh with 40 subintervals. The test was done for the sequence of non-uniform meshes described earlier.

Subintervals	2nd Avg Err	4th Avg Err	2nd Avg Err Est
20	0.0021	8.9311e-05	0.0020
40	5.4837e-04	3.5015e-06	5.4487e-04
80	1.3926e-04	6.0269e-08	1.3921e-04
160	3.506e-05	6.6100e-09	3.5065e-05
320	8.7942e-06	6.6022e-10	8.7949e-06
640	2.2021e-06	4.5985e-11	2.2021e-06

Table 5.2: Average errors for (4.1). The average error of the fourth order method is small enough in all cases to allow for a good error estimate. The test was done for the sequence of non-uniform meshes described earlier.

Subintervals	2nd Max Err	4th Max Err	2nd Max Err Est
20	0.0053	2.5301e-04	0.0051
40	0.0013	1.6283e-05	0.0013
80	3.3231e-04	7.0433e-07	3.3235e-4
160	8.3128e-05	2.5797e-08	8.3147e-5
320	2.0785e-05	1.8015e-09	2.0787e-5
640	5.1965e-06	1.2313e-10	5.1966e-6

Table 5.3: Max errors for (4.1). The fourth order method is able to provide a quality error even for the points with the maximum error. The test was done for the sequence of non-uniform meshes described earlier.

For (4.2) Table 5.4 shows that the error ratios of the second order finite difference scheme on a mesh of 20 subintervals compared to a mesh of 40 subintervals are far smaller than expected. Tables 5.5 and 5.6 show that this occurs due to unusually large error on the 20 subinterval mesh. This is due to the subinterval sizes being outside the asymptotic regime of the Taylor series. When the subintervals are outside the asymptotic regime of the Taylor's expansions, upon which the finite difference schemes are based, will not yield accurate results. When tested on the 40 subinterval mesh the subintervals were within the asymptotic regime, i.e. small enough, so the approximate solution on the 40 subinterval mesh was much more accurate than for the 20 subinterval mesh, leading to the low ratio. Subsequent tests show the second order finite difference scheme converging to $\frac{1}{2^2}$ which indicates second order convergence. The fourth order finite difference schemes ratios are approximately $\frac{1}{2^4}$, indicating fourth order convergence.

Tables 5.5 and 5.6 show that the error estimates are of good quality on all meshes, even for the case where the second order finite difference scheme is outside the asymptotic regime. This is because the fourth order finite difference scheme did not have a large error for this mesh. A graph for the exact solution of (4.2) can be seen in Figure 5.3.

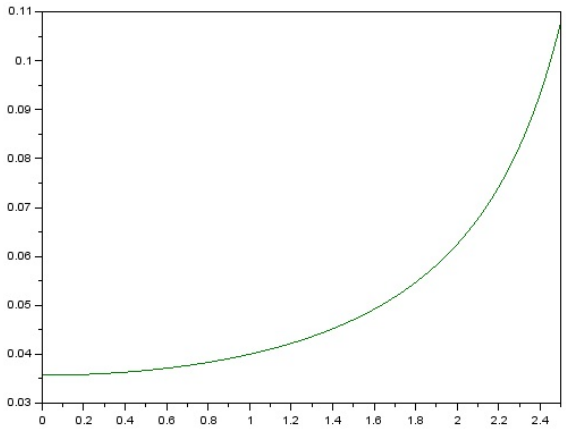

Figure 5.3: Graph of exact solution for (4.2).

Mesh Pairs	2nd Avg Ratio	2nd Max Ratio	4th Avg Ratio	4th Max Ratio
20/40	0.0023	9.0332e-4	0.1283	0.1774
40/80	0.2692	0.2650	0.0413	0.0713
80/160	0.2560	0.2538	0.0163	0.0494
160/320	0.2519	0.2510	0.2139	0.0390
320/640	0.2507	0.2502	0.0984	0.0490

Table 5.4: Convergence ratios for (4.2). The ratios are obtained by dividing the error of the numerical solution computed on the previous test by the error of the numerical solution computed for the current test. With 20 subintervals the sizes of the subintervals were outside of the asymptotic regime, leading to a large error and thus a very small 20/40 ratio. The test was done for the sequence of non-uniform meshes described earlier.

Subintervals	2nd Avg Err	4th Avg Err	Avg Err Est
20	0.1074	1.5318e-04	0.1073
40	2.4411e-4	1.9652e-5	2.2464e-04
80	6.5714e-5	8.1129e-7	6.4902e-05
160	1.6820e-5	1.3252e-8	1.6816e-05
320	4.2370e-06	2.8344e-9	4.2395e-06
640	1.0621e-06	2.7891e-10	1.0624e-06

Table 5.5: Average errors for (4.2). With 20 subintervals the second order finite difference scheme was outside the asymptotic regime while the fourth order method was within the asymptotic regime. This allows for a good quality error estimate as the accuracy of the fourth order finite difference scheme determines the accuracy of the error estimate. The test was done for the sequence of non-uniform meshes described earlier.

Subintervals	2nd Max Err	4th Max Err	Max Err Est
20	0.6432	4.7249e-04	0.6433
40	5.8101e-4	8.3805e-05	5.3668e-4
80	1.5398e-4	5.9723e-06	1.5255e-4
160	3.9079e-05	2.9480e-07	3.9079e-5
320	9.8108e-06	1.1494e-08	9.8161e-6
640	2.4551e-06	5.6313e-10	2.4557e-6

Table 5.6: Max errors for (4.2). With 20 subintervals the second order finite difference scheme was outside the asymptotic regime while the fourth order method was within the asymptotic regime. This allows for a good quality error estimate as the accuracy of the fourth order finite difference scheme determines the accuracy of the error estimate. The test was done for the sequence of non-uniform meshes described earlier.

5.2 BVODE Summary

The second and fourth order finite difference schemes both exhibit an experimental order of convergence consistent with theoretical order of convergence in all tests. The error estimate for the second order finite difference scheme agrees to at least one significant digit with the true error of the second order finite difference scheme in all tests.

5.3 1D PDE Results

The tests done in this section are cases where the finite difference schemes were applied to 1D PDEs to generate systems of IVODEs. Recall that the IVODEs were then passed to the Scilab *ode* function [6], which solved them using the lsoda solver. The finite difference schemes were applied to the x domain, and the same mesh as in Section 5.1 was used. For the computation performed by the Scilab *ode* function, a tolerance of 10^{-8} was chosen. This was to ensure the error associated with the solution of the IVODE systems did not significantly effect the error associated with the finite difference schemes. Error statistics were calculated as in Section 5.1.

Test problem (4.3) is the 1D Burger's equation [9] with initial and boundary conditions chosen so that the exact solution is a wave front that advances across the x domain. The problem can

be made easier or harder depending on the chosen value of ϵ. Here an ϵ value of 0.01 was chosen. A plot of the approximate solution using the fourth order finite difference scheme on a mesh of size 640 together with the Scilab *ode* function can be seen in Figure 5.4. Table 5.7 shows that for the second order finite difference scheme the ratios converge to $\frac{1}{2^2}$ as the size of the mesh increases, demonstrating second order convergence. The ratios for the fourth order finite difference scheme approach $\frac{1}{2^4}$ which demonstrates fourth order convergence. Tables 5.8 and 5.9 show that for both average and maximum errors the fourth order finite difference scheme was able to generate an error estimate of acceptable quality for the second order scheme, particularly as the mesh becomes finer.

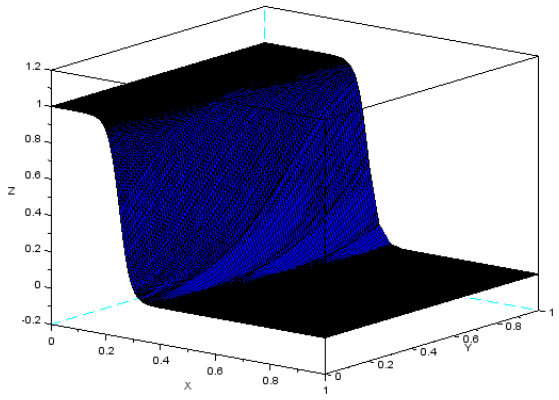

Figure 5.4: Graph of the fourth order approximation with 640 mesh points on domain $(t, x) \in [0, 1] \times [0, 1]$ for (4.3) with $\epsilon = 0.01$.

Mesh Pairs	2nd Avg Ratio	2nd Max Ratio	4th Avg Ratio	4th Max Ratio
40/80	0.2394	0.4916	0.1075	0.301
80/160	0.2589	0.3083	0.0957	0.1369
160/320	0.2507	0.2409	0.0769	0.0798
320/640	0.2502	0.2564	0.0695	0.0710

Table 5.7: Convergence ratios for (4.3). The ratios are obtained by dividing the error of the numerical solution computed on the previous mesh by the error of the numerical solution computed for the current mesh. The test was done for the sequence of non-uniform meshes described earlier.

Subintervals	2nd Avg Err	4th Avg Err	Avg Err Est
40	0.0040	0.0017	0.0026
80	9.6385e-4	1.8139e-4	8.2697e-4
160	2.4957e-4	1.7364e-5	2.3788e-4
320	6.2566e-5	1.3347e-6	6.1705e-5
640	1.5657e-5	9.2710e-8	1.5598e-5

Table 5.8: Average errors for (4.3). The quality of the error estimate improves significantly as the mesh size increases. The test was done for the sequence of non-uniform meshes described earlier.

Subintervals	2nd Max Err	4th Max Err	Max Err Est
40	0.1835	0.0950	0.0929
80	0.0902	0.0286	0.0616
160	0.0278	0.0039	0.0239
320	0.0067	3.1242e-4	0.0064
640	0.0017	2.2179e-5	0.0017

Table 5.9: Max errors for (4.3). The quality of the error estimate improves significantly as the mesh size increases. The test was done for the sequence of non-uniform meshes described earlier.

Test problem (4.4) has an exact solution that exhibits growth of a sinusoidal shape. A plot of the solution from the fourth order finite difference scheme on a mesh of size 640 can been seen in Figure 5.5. Table 5.10 shows that the second order finite difference scheme demonstrates the expected convergence rate. The fourth order finite difference scheme did not demonstrate the expected convergence rate, with the ratios being around 0.13 instead. This means that in this case the fourth order finite difference scheme showed an experimental rate of convergence closer to third order for (4.4). This is because the mesh is not adapted appropriately for the problem, and it is possible that with an appropriate mesh the finite difference scheme would exhibit a higher order of convergence. Tables 5.11 and 5.12 show that the fourth order scheme had larger average and maximum errors on the size 40 mesh as well as a larger average error on the size 80 mesh. This caused the schemes to significantly over estimate the error of the second order scheme on the first mesh. Despite this initial overestimate, the higher convergence rate of the fourth order method allowed for the error estimate to become of reasonable quality (within an order of magnitude) by the second mesh.

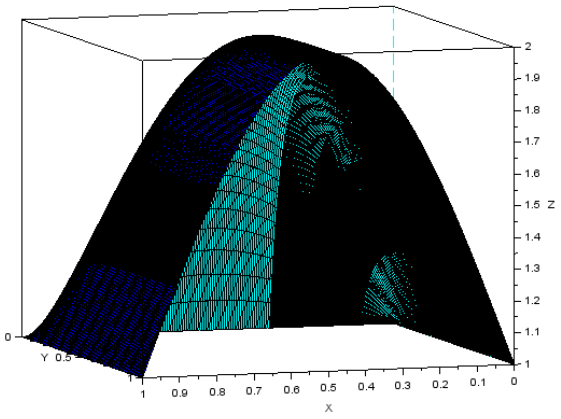

Figure 5.5: Plot of the approximate solution of (4.4) from the fourth order finite difference scheme on a mesh of size 640 on domain $(x, y) \in [0, 1] \times [0, 1]$.

Mesh Pairs	2nd Avg Ratio	2nd Max Ratio	4th Avg Ratio	4th Max Ratio
40/80	0.2518	0.2498	0.1237	0.1253
80/160	0.2522	0.2501	0.1272	0.1286
160/320	0.2537	0.2512	0.1205	0.1263
320/640	0.2587	0.2575	0.0960	0.1361

Table 5.10: Convergence ratios for (4.4). The ratios are obtained by dividing the error of the numerical solution computed on the previous mesh by the error of the numerical solution computed for the next mesh. The fourth order finite difference does not exhibit fourth order convergence on this problem. The test was done for the sequence of non-uniform meshes described earlier.

Subintervals	2nd Avg Err	4th Avg Err	Avg Err Est
40	4.9284e-4	0.0016	0.0021
80	1.2409e-4	2.0354e-4	3.2399e-4
160	3.1292e-5	2.5883e-5	5.6367e-5
320	7.9373e-6	3.1195e-6	1.0887e-5
640	2.0533e-6	2.9937e-7	2.2613e-6

Table 5.11: Average errors for (4.4). The error of the fourth order method was initially larger than that of the second order method, causing the error to be overestimated for tests on coarser meshes. The test was done for the sequence of non-uniform meshes described earlier.

Subintervals	2nd Max Err	4th Max Err	Max Err Est
40	0.0017	0.0034	0.0042
80	4.3055e-4	4.3001e-4	7.3020e-4
160	1.0768e-4	5.5306e-5	1.4553e-4
320	2.7046e-5	6.9862e-6	3.1607e-5
640	6.9641e-6	9.5067e-7	7.2940e-6

Table 5.12: Max errors for (4.4). The error of the fourth order method was initially larger than that of the second order method, causing the error to be overestimated for tests on coarser meshes. The test was done for the sequence of non-uniform meshes described earlier.

5.4 1D PDE Summary

For (4.3) both finite difference schemes exhibited experimental order of convergence consistent with theoretical order of convergence. Only the second order finite difference scheme demonstrated the expected order of convergence for (4.4). Error estimates were within one order of magnitude for all tests with (4.3). The error estimate for (4.4) was not within an order of magnitude on the mesh with 40 subintervals, and was within an order of magnitude for the meshes with 80 or more subintervals.

5.5 2D PDE Results

This section covers tests where the finite difference schemes are applied to 2D PDEs to generate a system of 1D PDEs which is then passed to BACOLI [9]. For these tests, the y variable was discretized by the finite difference schemes. This gave a system of 1D PDEs in terms of t and x that were passed to BACOLI. In order to be able to generate an error estimate the schemes

are applied to create two separate 1D PDE systems, one from the second order finite difference scheme discretization of the 2D PDE and one from the fourth order finite difference scheme discretization of the 2D PDE. These are passed as a single 1D PDE system to BACOLI. The meshes chosen for the y domain were chosen so that the mesh points would be clustered closer to the center of the domain rather than the boundaries. This gives [0, 0.25, 0.45, 0.55, 0.75, 1.0] for the 5 subinterval non-uniform mesh, [0, 0.145, 0.310, 0.425, 0.490, 0.500, 0.510, 0.575, 0.690, 0.855, 1.0] for the 10 subinterval non-uniform mesh, and [0, 0.063, 0.146, 0.220, 0.286, 0.343, 0.393, 0.433, 0.465, 0.490, 0.500, 0.510, 0.534, 0.567, 0.608, 0.657, 0.714, 0.780, 0.854, 0.937, 1.0] for the 20 subinterval non-uniform mesh that we employ in our experiments. Uniform meshes of the same sizes were also considered. The above non-uniform meshes were chosen arbitrarily and used in all tests. If the meshes were adapted to the problems and continued to adapt as the solution progressed then the errors on the non-uniform meshes would be smaller than on the uniform meshes [9].

5.5.1 Test Problem (4.5) with Neumann Boundary Conditions

Test problem (4.5) is the 2D Burger's Equation [12]. Similar to the 1D Burger's Equation this describes a 2D wavefront moving across the domain. The difficulty of this problem is determined by ϵ, with an ϵ of 0.1 being chosen in our tests. This problem tests the finite difference schemes on their ability to approximate the first and second derivatives of the y variable.

The ratios in Table 5.13 show that both schemes demonstrate an experimental order of convergence consistent with theoretical order of convergence across the uniform mesh tests when using Neumann boundary conditions. The second order ratios are around $\frac{1}{2^2}$ and the fourth order ratios are around $\frac{1}{2^4}$. Both methods also begin to show the expected order of convergence on the non-uniform meshes once the mesh size increases. Tables 5.14 and 5.15 show a good quality error estimate that gives the correct order of magnitude. The error estimate is also of good quality on non-uniform meshes, again giving an error estimate that is of the correct order of magnitude.

All meshes are relatively coarse to deal with limited computing space on the server we used for the testing. The non-uniform meshes were chosen arbitrarily so that the same mesh could be used in both tests. Using adapted meshes and further adapting the meshes over the course of the approximation would lead to the errors on the non-uniform meshes being smaller than for the uniform meshes [9]. In its current state, the software does not support these features for the y domain, though BACOLI does do this in the x domain [7]. Figure 5.7 shows that the error on the 20 subinterval uniform mesh was distributed across the wave front with errors largest at the corners (1,0,1) and (0,1,1). Figure 5.8 shows that on the non-uniform mesh the error is larger

on the left half of the y domain, suggesting a non-uniform mesh with most points clustered on that half could produce better results. A solution on the 20 subinterval uniform mesh at $t = 1$ can be seen in Figure 5.6.

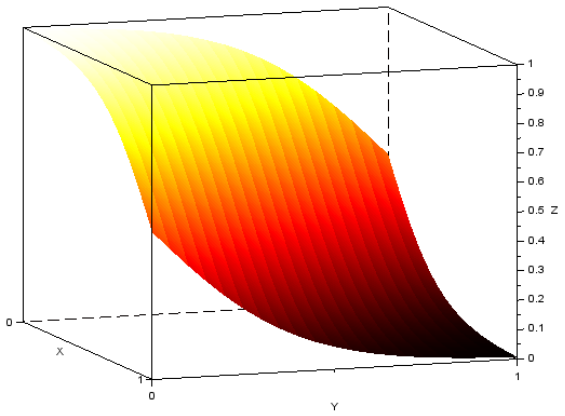

Figure 5.6: Plot of the approximate solution of (4.5) on a uniform mesh of size 20 using the fourth order scheme with Neumann boundary conditions on domain $(x, y) \in [0, 1] \times [0, 1]$ at $t = 1$. Discretization applied to y domain.

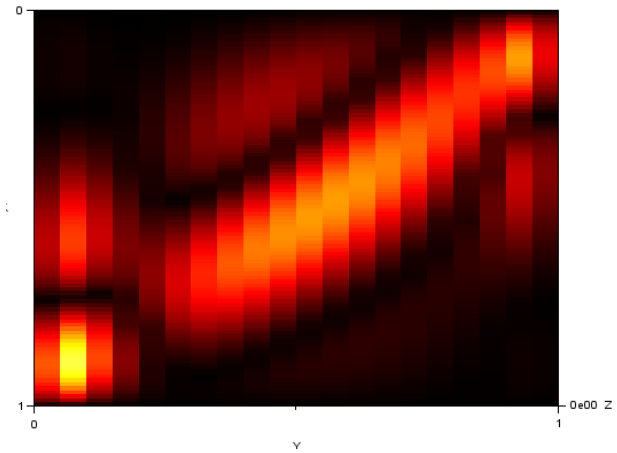

Figure 5.7: Error of the approximate solution of (4.5) on a uniform mesh of size 20 using the fourth order scheme with Neumann boundary conditions on domain $(x, y) \in [0, 1] \times [0, 1]$ at $t = 1$. Lighter color means larger error. Discretization applied to y domain.

Mesh Pairs	2nd Avg Ratio	2nd Max Ratio	4th Avg Ratio	4th Max Ratio
5/10 uni	0.2787	0.2540	0.0476	0.0461
10/20 uni	0.2631	0.2487	0.0520	0.0366
5/10 non	0.4593	0.4307	0.1108	0.1404
10/20 non	0.2214	0.2120	0.0346	0.0244

Table 5.13: Convergence ratios for (4.5) with Neumann boundary conditions. The ratios are obtained by dividing the error of the numerical solution computed on the previous mesh by the error of the numerical solution computed for the current mesh. The test was done on uniform and non-uniform meshes described earlier.

Subintervals	2nd Avg Err	4th Avg Err	Avg Err Est
5 uni	0.0033	0.0010	0.0027
10 uni	9.2878e-4	4.7666e-5	8.8233e-4
20 uni	2.4432e-4	2.4784e-6	2.4186e-4
5 non	0.0046	0.0017	0.0032
10 non	0.0021	1.9314e-4	0.0019
20 non	4.6385e-4	6.6788e-6	4.5729e-4

Table 5.14: Average errors for (4.5) with Neumann boundary conditions. The error estimate is within an order of magnitude of the true error using as few as 5 subintervals. The test was done on uniform and non-uniform meshes described earlier. Discretization applied to y domain.

Subintervals	2nd Max Err	4th Max Err	Max Err Est
5 uni	0.0138	0.0091	0.0134
10 uni	0.0035	4.1893e-4	0.0034
20 uni	8.7162e-4	1.5331e-5	8.6222e-4
5 non	0.0193	0.0152	0.0144
10 non	0.0083	0.0021	0.0080
20 non	0.0018	5.2140e-5	0.0017

Table 5.15: Max errors for (4.5) with Neumann boundary conditions. The error estimate is within an order of magnitude of the true error using as few as 5 subintervals. The test was done on uniform and non-uniform meshes described earlier. Discretization applied to y domain.

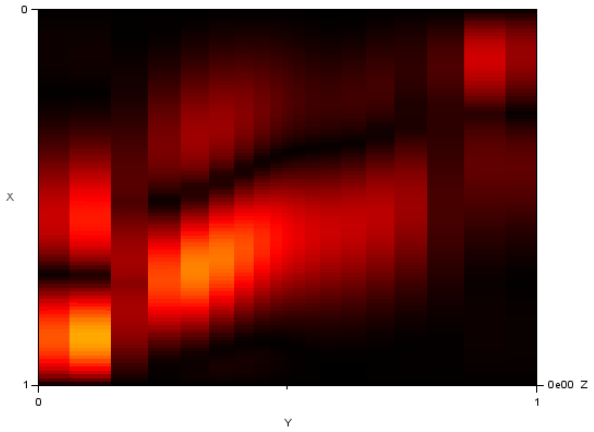

Figure 5.8: Plot of the approximate solution of (4.5) on a non-uniform mesh of size 20 using the fourth order scheme with Neumann boundary conditions.

5.5.2 Test Problem (4.5) with Dirichlet Boundary Conditions

Since (4.5) does not have a cross derivative term involving u_{xy}, the boundary conditions for this problem are entirely Dirichlet. Table 5.16 shows that the method provided virtually identical ratios for the uniform meshes as it did with the Neumann boundary conditions in Table 5.13. The fourth order finite difference scheme also produced identical results for the non-uniform meshes with both styles of boundary conditions. The difference between the two methods is in the second order scheme. On the non-uniform mesh with 10 subintervals the error decreased by only a very small amount. Tables 5.17 and 5.18 show again that the different methods of

defining the boundary conditions lead to almost identical results except in the case where the second order scheme is used with non-uniform meshes. Both boundary condition approaches produced identical results on the size 5 non-uniform mesh but the Dirichlet boundary conditions had significantly larger error on the size 10 and 20 non-uniform mesh. Since the fourth order finite difference scheme did not experience this effect the quality of the error estimate was not effected.

Mesh Pairs	2nd Avg Ratio	2nd Max Ratio	4th Avg Ratio	4th Max Ratio
5/10 uni	0.2787	0.2540	0.0476	0.0461
10/20 uni	0.2631	0.2487	0.0521	0.0365
5/10 non	0.9289	2.6528	0.1108	0.1404
10/20 non	0.2412	0.3321	0.0346	0.0244

Table 5.16: Convergence ratios for (4.5) with Dirichlet bounds. The ratios are obtained by dividing the error of the numerical solution computed on the previous mesh by the error of the numerical solution computed for the current mesh. The test was done on the uniform and non-uniform meshes listed previously.

Subintervals	2nd Avg Err	4th Avg Err	Avg Err Est
5 uni	0.0033	0.0010	0.0027
10 uni	9.2878e-4	4.7665e-5	8.8233e-4
20 uni	2.4432e-4	2.4818e-6	2.4186e-4
5 non	0.0046	0.0017	0.0032
10 non	0.0042	1.9313e-4	0.0040
20 non	0.0010	6.6784e-6	0.0010

Table 5.17: Average errors for (4.5) with Dirichlet bounds. The mesh placement of the size 10 non-uniform mesh leads to unusually high error. The test was done on the uniform and non-uniform meshes listed previously.

Subintervals	2nd Max Err	4th Max Err	Max Err Est
5 uni	0.0138	0.0091	0.0134
10 uni	0.0035	4.1892e-4	0.0034
20 uni	8.7162e-4	1.5291e-5	8.6222e-4
5 non	0.0193	0.0152	0.0144
10 non	0.0511	0.0021	0.0508
20 non	0.0170	5.2094e-5	0.0169

Table 5.18: Max errors for (4.5) with Dirichlet bounds. The mesh placement of the size 10 non-uniform mesh leads to unusually high error. The test was done on the uniform and non-uniform meshes listed previously.

5.5.3 Test Problem (4.6) with Neumann Boundary Conditions

Test problem (4.6) tests the methods on a problem with a cross derivative term, u_{xy}. Recall y is the variable being discretized, so in this case the method is applied as though this is a first derivative. This produces a system involving the derivative u_x which can be passed to

BACOLI. Test problem (4.6) describes a sinusoidal shape shrinking over time. A solution for the 20 subinterval uniform mesh at $t = 1$ can be seen in Figure 5.9.

Table 5.19 shows that both the second order and fourth order finite difference schemes demonstrated unusual convergence rates on the uniform meshes. However, in the case of non-uniform meshes both schemes had an experimental rate of convergence consistent with their theoretical rates. Tables 5.20 and 5.21 show that the issue with the uniform meshes was on the mesh with 20 subintervals. **Figure 5.10 shows that the scheme had the largest error near the center of the domain, indicating the need for a different placement of the points. The non-uniform mesh clustered the points closer to the center where the error was largest, and the effect this had can be seen in Figure 5.11 where the error is now evenly distributed across the domain.** The error estimate associated with the computations on non-uniform meshes was of good quality in all cases, being within an order of magnitude of the true error. The error estimate was of good quality for size 5 and 10 uniform meshes, but was not within an order of magnitude for the size 20 uniform mesh.

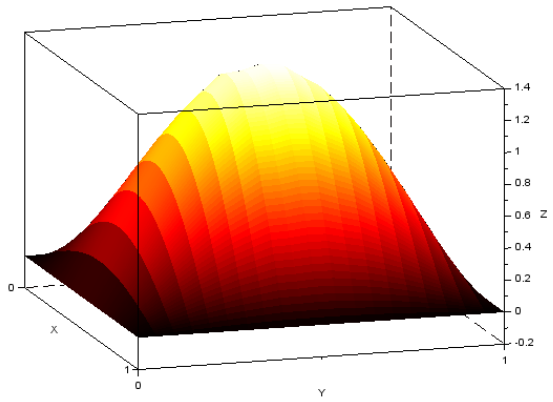

Figure 5.9: Plot of approximate solution of (4.6) on a uniform mesh with 20 subintervals using the fourth order scheme with Neumann boundary conditions at $t = 1$.

Mesh Pairs	2nd Avg Ratio	2nd Max Ratio	4th Avg Ratio	4th Max Ratio
5/10 uni	0.2834	0.2477	0.0798	0.0859
10/20 uni	2.0258	8.5239	40.5823	119.16
5/10 non	0.4514	0.4380	0.0989	0.2181
10/20 non	0.2473	0.2251	0.0737	0.0290

Table 5.19: Convergence ratios for (4.6) with Neumann bounds. The ratios are obtained by dividing the error of the numerical solution computed on the previous mesh by the error of the numerical solution computed for the current mesh. The test was done on uniform and non-uniform meshes described earlier.

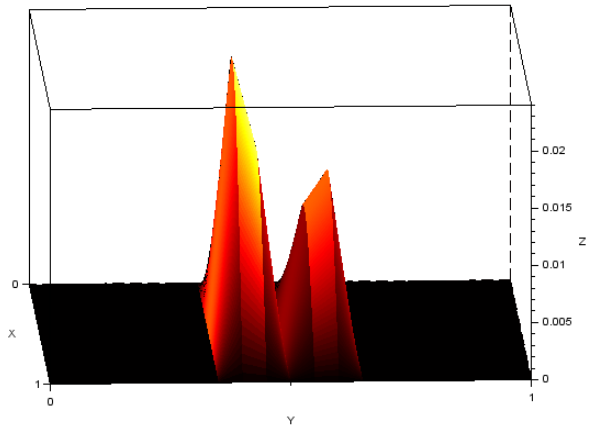

Figure 5.10: Error of approximate solution of (4.6) on a uniform mesh with 20 subintervals using the fourth order scheme with Neumann boundary conditions at $t = 1$. There is significant error in the center of the domain.

Subintervals	2nd Avg Err	4th Avg Err	Avg Err Est
5 uni	0.0037	6.0116e-4	0.0031
10 uni	0.0010	4.7949e-5	0.0010
20 uni	0.0021	0.0019	2.1550e-4
5 non	0.0078	9.1818e-4	0.0069
10 non	0.0035	9.0843e-5	0.0034
20 non	8.6991e-4	6.6955e-6	8.6329e-4

Table 5.20: Average errors for (4.6) with Neumann bounds. Both schemes had trouble with the uniform mesh of size 20 leading to a low quality solution and error estimate. The test was done on uniform and non-uniform meshes described earlier.

Subintervals	2nd Max Err	4th Max Err	Max Err Est
5 uni	0.0116	0.0023	0.0107
10 uni	0.0029	2.0120e-4	0.0028
20 uni	0.0244	0.0240	5.6433e-4
5 non	0.0222	0.0034	0.0188
10 non	0.0097	7.4673e-4	0.0096
20 non	0.0022	2.1639e-5	0.0022

Table 5.21: Max errors for (4.6) with Neumann bounds. Both schemes had trouble with the uniform mesh of size 20 leading to a low quality solution and error estimate. The test was done on uniform and non-uniform meshes described earlier.

5.5.4 Test Problem (4.6) with Dirichlet bounds

Since (4.6) contains the u_{xy} cross derivative our schemes need boundary information for u_x in addition to the Dirichlet boundary conditions. Table 5.22 shows that when using this type of boundary condition both finite difference schemes exhibited experimental convergence rates consistent with theoretical convergence rates on the uniform meshes. The fourth order finite difference scheme had identical ratios for the non-uniform meshes as in the Neumann boundary conditions test, but this time the second order finite difference scheme had larger error on the

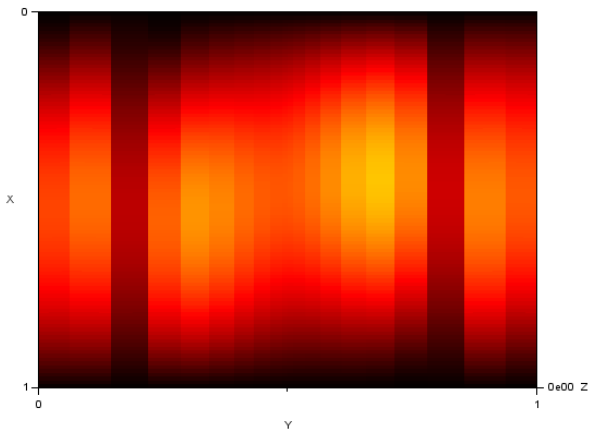

Figure 5.11: Error of approximate solution of 5.19 on the non-uniform mesh with 20 subintervals described earlier using the fourth order scheme with Neumann boundary conditions at $t = 1$. The error is fairly evenly distributed.

non-uniform meshes. Tables 5.23 and 5.24 show that a good quality error estimate in is obtained all cases, with error estimates exhibiting correct order of magnitude.

Mesh Pairs	2nd Avg Ratio	2nd Max Ratio	4th Avg Ratio	4th Max Ratio
5/10 uni	0.2834	0.2477	0.0798	0.0859
10/20 uni	0.2655	0.2526	0.0414	0.0448
5/10 non	2.2093	3.4913	0.0989	0.2181
10/20 non	0.4791	0.5999	0.0737	0.0290

Table 5.22: Convergence ratios for (4.6) with Dirichlet bounds. The ratios are obtained by dividing the error of the numerical solution computed on the previous mesh by the error of the numerical solution computed for the next mesh. The test was done on uniform and non-uniform meshes described earlier.

Subintervals	2nd Avg Err	4th Avg Err	Avg Err Est
5 uni	0.0037	6.0116e-4	0.0031
10 uni	0.0010	4.7949e-5	0.0010
20 uni	2.7825e-4	1.9862e-6	2.7629e-4
5 non	0.0101	9.1818e-4	0.0091
10 non	0.0222	9.0843e-5	0.0221
20 non	0.0106	6.6950e-6	0.0106

Table 5.23: Average errors for (4.6) with Dirichlet bounds. The second order finite difference schemes error does not decrease as more points are added in the non-uniform case. The test was done on uniform and non-uniform meshes described earlier.

5.6 2D PDE Neumann Summary

Both finite difference schemes exhibited experimental order of convergence consistent with theoretical order of convergence except on the uniform mesh with 20 subintervals for (4.6). This is because the placement of mesh points with the uniform mesh leads to very large errors in the

Subintervals	2nd Max Err	4th Max Err	Max Err Est
5 uni	0.0116	0.0023	0.0107
10 uni	0.0029	2.0120e-4	0.0028
20 uni	7.2307e-4	9.0216e-6	7.2298e-4
5 non	0.0385	0.0034	0.0359
10 non	0.1346	7.4673e-4	0.1338
20 non	0.0807	2.1639e-5	0.0807

Table 5.24: Max errors for (4.6) with Dirichlet bounds. The second order finite difference schemes error does not decrease as more points are added in the non-uniform case. The test was done on uniform and non-uniform meshes described earlier.

center of the (x, y) domain for this problem. This was remedied using a non-uniform mesh. The error estimate for the second order finite difference scheme was within one order of magnitude of the true error for all cases except for the mesh with 20 subintervals for (4.6).

5.7 2D PDE Dirichlet Summary

The fourth order finite difference scheme exhibited experimental order of convergence consistent with theoretical order of convergence in all cases. The second order finite difference scheme exhibited experimental order of convergence consistent with theoretical order of convergence on all uniform meshes. The second order finite difference scheme did not exhibit an experimental order of convergence consistent with theoretical order of convergence on non-uniform meshes. The error estimates were within an order of magnitude of the true error for the second order finite difference scheme in all cases.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

Both finite difference schemes demonstrated experimental orders of convergence consistent with their theoretical orders of convergence in all cases for BVODE tests. This led to the fourth order finite difference scheme having a significantly more accurate approximation, and thus the error estimates for the second order finite difference schemes are all within an order of magnitude of the true error. Thus these finite difference schemes can be used together to create numerical software with adaptive mesh refinement and error control for BVODEs.

For 1D PDEs, the second order and fourth order finite difference scheme demonstrated the expected order of convergence for (4.3). Only the second order finite difference scheme demonstrated the expected order of convergence for (4.4). The error estimates were all within an order of magnitude of the true error for the second order finite difference scheme for (4.3). The error estimates were not of good quality on coarser meshes for (4.4) due to the lower order of convergence exhibited by the fourth order finite difference scheme in this test. Once a finer mesh was employed the error estimates were once again of good quality, so these finite difference schemes can be used together to create numerical software with adaptive mesh refinement and error control for 1D PDEs.

When solving 2D PDEs with Neumann boundary conditions, both finite difference schemes tended to show experimental orders of convergence approaching the theoretical orders of convergence as the mesh became finer. For problem (4.6) the placement of mesh points of the uniform mesh with 20 subintervals caused significantly increased error, but this increased error was remedied using a non uniform mesh of the same size. In all cases, except the previously mentioned case, the error estimate was within an order of magnitude of the true error of the second order finite difference scheme.

When solving 2D PDEs with Dirichlet boundary conditions, the fourth order finite difference scheme exhibited the expected order of convergence in all cases. On uniform meshes, the second order finite difference scheme demonstrated the expected order of convergence. On non-uniform meshes, the second order finite difference scheme did not exhibit an experimental order of convergence consistent with theoretical order of convergence. Since the fourth order scheme did not exhibit the same issue, the error estimates were of good quality in all cases.

Both methods for treating the boundary conditions produced good quality error estimates, but the second order finite difference scheme has significantly larger error on non-uniform meshes when Dirichlet boundary conditions rather then Neumann boundary conditions were used. Thus we conclude that when solving 2D PDEs using these finite difference schemes, Neumann boundary conditions should be used.

6.2 Future Work

Repeat some of the computations using meshes that are appropriately adapted to the solution behavior. This would hopefully improve the convergence results in the cases where this thesis has identified issues.

Compute error ratios in a point wise sense. It is possible some of the issues with error ratios stem from the way the ratios were calculated. Recall that for average error ratios, means were computed of the exact errors for each mesh used for testing, and then the ratio was calculated by dividing the mean error of one mesh by the mean error of a mesh with subintervals that were half of the size. Maximum error ratios were calculated by dividing the maximum error of one mesh by the maximum error of a mesh with subintervals that were half of the size, even if these occurred at different points in each mesh. The convergence results may be more consistent if the average ratios were calculated by computing the error ratio of the all points shared between the two meshes, and then taking the mean of those ratios. The maximum error ratio should be determined as the ratio between the point with the most error in the coarser mesh and the same point in the finer mesh.

Optimization of the fourth order finite difference scheme would increase the speed at which the software can compute solutions. In the current implementation, there are large numbers of repeated multiplications and additions when calculating the coefficients for the fourth order finite difference scheme that slow the process down. Pre-calculating and saving the more common sub-expressions in each coefficient would help to mitigate this effect.

This thesis shows that the fourth order finite difference scheme is able to produce a quality error estimate for the second order finite difference scheme for BVODEs, as well as 1D and 2D

PDEs. Thus, these schemes could be used to create numerical software with error estimation and control similar to the software in [9]. The software should use this error estimate to refine the mesh to evenly distribute error across the whole domain. This allows the software to meet a user tolerance using as few mesh points as possible, leading to fewer calculations in total and thus more efficient software.

Bibliography

[1] Thomas Hillen, Kevin J. Painter, and Michael Winkler. Anisotropic diffusion in oriented environments can lead to singularity formation. *European Journal of Applied Mathematics*, 24(3):371–413, 2013.

[2] Kristin R. Swanson, Ellsworth C. Alvord, Jr., and James. D. Murray. Dynamics of a model for brain tumors reveals a small window for therapeutic intervention. *Discrete and Continuous Dynamical Systems. Series B*, 4(1):289–295, 2004.

[3] Nikolaos Sfakianakis. *Finite difference schemes on non-uniform meshes for hyperbolic conservation laws*. PhD thesis. University of Crete, 2009.

[4] Bengt Fornberg. Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of Computation*, 51(184):699–706, 1988.

[5] Scilab fsolve function. `https://help.scilab.org/docs/6.0.0/en_US/fsolve.html`. Accessed: 2016-10-06.

[6] Scilab ode function. `https://help.scilab.org/docs/6.0.0/en_US/ode.html`. Accessed: 2016-10-15.

[7] Tom Arsenault, Tristan Smith, Paul Muir, and Pat Keast. Efficient interpolation-based error estimation for 1D time-dependent PDE collocation codes. URL `http://cs.smu.ca/tech_reports/txt2011_001.pdf`. Department of Mathematics and Computer Science, Saint Mary's University, Technical Report 2011-001, 2011.

[8] Alexander H.-D. Cheng and Daisy T. Cheng. Heritage and early history of the boundary element method. *Engineering Analysis with Boundary Elements*, 29(3):268 – 302, 2005.

[9] Jack Pew, Zhi Li, and Paul Muir. Algorithm 962: BACOLI: B-spline adaptive collocation software for PDEs with interpolation-based spatial error control. *ACM Transactions on Mathematical Software*, 42(3):25:1–25:17, 2016.

[10] Scilab. `http://www.scilab.org/scilab/about`. Accessed: 2017-05-09.

[11] Sympy. http://www.sympy.org/en/index.html. Accessed: 2016-09-13.

[12] Zhi Li. B-spline collocation for two dimensional, time-dependent, parabolic PDEs. Master's thesis. Department of Mathematics and Computing Science, Saint Mary's University, 2012.

Appendix A

Source Code

Both styles of boundary conditions require that the 2D PDEs are in the form

$$u_t = f(t, x, y, u, u_x, u_y, u_{xx}, u_{xy}, u_{yy})$$

The software then converts this into a system of 1D PDEs in the form

$$u_t = f(t, x, u, u_x, u_{xx})$$

that can be passed to BACOLI. For usage of BACOLI refer to [7]. All code in this section is such that y is the variable being discretized by the finite difference scheme and x and t are dealt with by BACOLI.

A.1 Problem Definition (4.5) with u_t Style Bounds

The following code is the definition of (4.5) using Neumann boundary conditions. Both the PDE and the boundary conditions for the variable being discretized, y, are defined in the function fusererrrest. Usage is that fval(1) and fval(n) are the boundary conditions for the fourth order finite difference scheme, with fval(2) to fval(n-1) being the definition of the PDE itself that will be used with the fourth order scheme. fval(n+1) and fval(2n) are the boundary conditions for the second order finite difference scheme, with fval(n+2) to fval(2n-1) similarly being the definition of the PDE to be used with the second order scheme. Note that the boundary conditions for the variable being discretized, y, are defined in the same function similar to the PDE.

```
c       This is the subroutine wherre the user defines their problem
c       in 2D, which is then translated into 1D PDEs and passed to f
c       so that bacoli can handle them. This version produces an error
```

```
c       estimate.
c-----------------------------------------------------------------------
      subroutine fusererrest(t, x, u, ux, uy, uxy, uyy, uxx, fval, npde)

c-----------------------------------------------------------------------
c purpose:
c       this subroutine defines the right hand side vector of the
c       npde dimensional parabolic partial differential equation
c                       ut = f(t, x, u, ux, uy, uxy, uyy, uxx).
c
c-----------------------------------------------------------------------
c subroutine parameters:
c input:
        integer                 npde
c                               the number of pdes in the system.
c
        double precision        t
c                               the current time coordinate.
c
        double precision        x
c                               the current spatial coordinate.
c
        double precision        u(npde)
c                               u(1:npde) is the approximation of the
c                               solution at the point (t,x,y).
c
        double precision        ux(npde)
c                               ux(1:npde) is the approximation of the
c                               spatial derivative in x of the solution at
c                               the point (t,x,y).

        double precision        uy(npde)
c                               uy(1:npde) is the approximation of the
c                               spatial derivative in x of the solution at
c                               the point (t,x,y).
```

```
c
        double precision        uxy(npde)
c                               uxy(1:npde) is the approximation of the
c                               cross spatial derivative of the
c                               solution at the point (t,x,y).

c
        double precision        uyy(npde)
c                               uyy(1:npde) is the approximation of the
c                               second spatial derivative in y of the
c                               solution at the point (t,x,y).

        double precision        uxx(npde)
c                               uxx(1:npde) is the approximation of the
c                               second spatial derivative in x of the
c                               solution at the point (t,x,y).
c
c output:
        double precision        fval(npde)
c                               fval(1:npde) is the right hand side
c                               vector f(t, x, u, ux, uxx) of the pde.
c-----------------------------------------------------------------------
        integer ymax
        parameter (ymax = 80)
        double precision a, b, w1, w2, z1, z2, l, gamma, eta
        common /tumor/ a, b, w1, w2, z1, z2, l, gamma, eta
        double precision xp(5), yp(5), yvals(ymax), e
        common /tumor/ xp, yp, yvals, e
        integer tumNum
        common /tumor/ tumNum
c-----------------------------------------------------------------------
c
c local variables
        double precision        D, Dx, Dy
c-----------------------------------------------------------------------
```

```
c loop indices:
        integer                 i, j, n
c-----------------------------------------------------------------------


      n = npde/2
      fval(1) = e**((yvals(1) + x - t)/(2*eta))
     +        /(2*eta*(1+e**((yvals(1) + x - t)/(2*eta)))**2)
      fval(n+1) = e**((yvals(1) + x - t)/(2*eta))
     +        /(2*eta*(1+e**((yvals(1) + x - t)/(2*eta)))**2)


      do i=2,(n-1)


        fval(i) = eta*uxx(i) + eta*uyy(i) - u(i)*(ux(i) + uy(i))
        fval(n+i) = eta*uxx(n+i) + eta*uyy(n+i) - u(n+i)*(ux(n+i)
     +          + uy(n+i))


      end do


      fval(n) = e**((yvals(n) + x - t)/(2*eta))
     +        /(2*eta*(1+e**((yvals(n) + x - t)/(2*eta)))**2)
      fval(2*n) = e**((yvals(n) + x - t)/(2*eta))
     +        /(2*eta*(1+e**((yvals(n) + x - t)/(2*eta)))**2)


      return
      end
```

The boundary conditions for x are defined in the way BACOLI requires. This is done in two functions, bndxa for the left x boundary condition and bndxb for the right x boundary condition. The left boundary conditions are defined as

```
      subroutine bndxa(t, u, ux, bval, npde)
c-----------------------------------------------------------------------
c purpose:
c       the subroutine is used to define the boundary conditions at the
c       left spatial end point x = xa.
c                           b(t, u, ux) = 0
c
```

```
c-----------------------------------------------------------------------
c subroutine parameters:
c input:
        integer                 npde
c                               the number of pdes in the system.
c
        double precision        t
c                               the current time coordinate.
c
        double precision        u(npde)
c                               u(1:npde) is the approximation of the
c                               solution at the point (t,xa).
c
        double precision        ux(npde)
c                               ux(1:npde) is the approximation of the
c                               spatial derivative of the solution at
c                               the point (t,xa).
c
c output:
        double precision        bval(npde)
c                               bval(1:npde) is the boundary condition
c                               at the left boundary point.
c-----------------------------------------------------------------------
        integer ymax
        parameter (ymax = 80)
        double precision a, b, w1, w2, z1, z2, l, gamma, eta
        common /tumor/ a, b, w1, w2, z1, z2, l, gamma, eta
        double precision xp(5), yp(5), yvals(ymax), e
        common /tumor/ xp, yp, yvals, e
        integer tumNum
        common /tumor/ tumNum
c-----------------------------------------------------------------------
c loop indices:
        integer                 i, n
c-----------------------------------------------------------------------
```

```
        n = npde/2
      do i = 1, n
         bval(i) = -1/(1+e**( (yvals(i) + a - t)/(2*eta) ) ) + u(i)
         bval(n + i) = -1/(1+e**( (yvals(i) + a - t)/(2*eta) ) )+ u(n+i)
      end do


c
      return
      end
```

The right boundary conditions are defined within the function bndxb and for (4.5) are

```
      subroutine bndxb(t, u, ux, bval, npde)
c-----------------------------------------------------------------------
c purpose:
c       the subroutine is used to define the boundary conditions at the
c       right spatial end point x = xb.
c                           b(t, u, ux) = 0
c
c-----------------------------------------------------------------------
c subroutine parameters:
c input:
        integer                 npde
c                               the number of pdes in the system.
c
        double precision        t
c                               the current time coordinate.
c
        double precision        u(npde)
c                               u(1:npde) is the approximation of the
c                               solution at the point (t,xb).
c
        double precision        ux(npde)
c                               ux(1:npde) is the approximation of the
c                               spatial derivative of the solution at
c                               the point (t,xb).
c
```

```
c output:
        double precision        bval(npde)
c                               bval(1:npde) is the boundary condition
c                               at the right boundary point.
c-----------------------------------------------------------------------
        integer ymax
        parameter (ymax = 80)
        double precision a, b, w1, w2, z1, z2, l, gamma, eta
        common /tumor/ a, b, w1, w2, z1, z2, l, gamma, eta
        double precision xp(5), yp(5), yvals(ymax), e
        common /tumor/ xp, yp, yvals, e
        integer tumNum
        common /tumor/ tumNum
c-----------------------------------------------------------------------
c loop indices:
        integer                 i, n
c-----------------------------------------------------------------------
        n = npde/2
      do i = 1, n
         bval(i) = -1/(1+e**( (yvals(i) + b - t)/(2*eta) ) ) + u(i)
         bval(n + i) = -1/(1+e**( (yvals(i) + b - t)/(2*eta) ) )+ u(n+i)


      end do
c
      return
      end
```

Initial conditions for the PDE are defined in the function uiniterrest. The information must be in the form

$$u = f(t_i, x, y)$$

where t_i is the initial value in the t variable domain. This allows the finite difference schemes to be used with the method of lines approach with BACOLI.

```
      subroutine uiniterrest(x, u, npde)
c-----------------------------------------------------------------------
c purpose:
```

```
c       this subroutine is used to return the npde-vector of initial
c       conditions of the unknown function at the initial time t = t0
c       at the spatial coordinate x.
c
c-----------------------------------------------------------------------
c subroutine parameters:
c input:
        double precision        x
c                               the spatial coordinate.
c
        integer                 npde
c                               the number of pdes in the system.
c
c output:
        double precision        u(npde)
c                               u(1:npde) is vector of initial values of
c                               the unknown function at t = t0 and the
c                               given value of x.
c-----------------------------------------------------------------------
      double precision pi
      parameter       (pi = 3.14159265358979323846264d0)
c-----------------------------------------------------------------------
        integer ymax
        parameter (ymax = 80)
        double precision a, b, w1, w2, z1, z2, l, gamma, eta
        common /tumor/ a, b, w1, w2, z1, z2, l, gamma, eta
        double precision xp(5), yp(5), yvals(ymax),e
        common /tumor/ xp, yp, yvals,e
        integer tumNum
        common /tumor/ tumNum
c-----------------------------------------------------------------------
c loop indices:
        integer                 i
c-----------------------------------------------------------------------
c local variables
```

```
        double precision        temp
c-----------------------------------------------------------------------
c
c     assign u(1:npde) the initial values of u(t0,x).
c
      do i = 1, npde/2
         u(i) = 1/(1+e**( (yvals(i) + x)/(2*eta) ) )
         u(npde/2 + i) = 1/(1+e**( (yvals(i) + x)/(2*eta) ) )
      end do
c
      return
      end
```

A.2 Problem Definition (4.6) with Dirichlet Bounds

The main difference to note between the approaches to boundary conditions is that with Dirichlet bounds the boundary conditions of the variable being discretized are not defined within fusererrest. Instead only the definition of the PDE is put within fusererrest.

```
      subroutine fusererrest(t, x, u, ux, uy, uxy, uyy, uxx, fval, npde)


c-----------------------------------------------------------------------
c purpose:
c       this subroutine defines the right hand side vector of the
c       npde dimensional parabolic partial differential equation
c                        ut = f(t, x, u, ux, uy, uxy, uyy, uxx).
c
c-----------------------------------------------------------------------
c subroutine parameters:
c input:
        integer                 npde
c                               the number of pdes in the system.
c
        double precision        t
c                               the current time coordinate.
c
```

```
      double precision        x
c                             the current spatial coordinate.
c
      double precision        u(npde)
c                             u(1:npde) is the approximation of the
c                             solution at the point (t,x,y).
c
      double precision        ux(npde)
c                             ux(1:npde) is the approximation of the
c                             spatial derivative in x of the solution at
c                             the point (t,x,y).

      double precision        uy(npde)
c                             uy(1:npde) is the approximation of the
c                             spatial derivative in x of the solution at
c                             the point (t,x,y).
c
      double precision        uxy(npde)
c                             uxy(1:npde) is the approximation of the
c                             cross spatial derivative of the
c                             solution at the point (t,x,y).

c
      double precision        uyy(npde)
c                             uyy(1:npde) is the approximation of the
c                             second spatial derivative in y of the
c                             solution at the point (t,x,y).

      double precision        uxx(npde)
c                             uxx(1:npde) is the approximation of the
c                             second spatial derivative in x of the
c                             solution at the point (t,x,y).
c
c output:
      double precision        fval(npde)
```

```
c                               fval(1:npde) is the right hand side
c                               vector f(t, x, u, ux, uxx) of the pde.
c-----------------------------------------------------------------------
        integer ymax
        parameter (ymax = 80)
        double precision a, b, w1, w2, z1, z2, l, gamma, eta
        common /tumor/ a, b, w1, w2, z1, z2, l, gamma, eta
        double precision xp(5), yp(5), yvals(ymax), e
        common /tumor/ xp, yp, yvals, e
        integer tumNum
        common /tumor/ tumNum
c-----------------------------------------------------------------------
c
c local variables
        double precision        L1, L2, L3, hold, y, pi
c-----------------------------------------------------------------------
c loop indices:
        integer                 i, j, n
c-----------------------------------------------------------------------

      n = npde/2
      do i=1,(n)

        call alt(t,x,yvals(i+1), hold)
        y = yvals(i+1)

        L1 = (x**2 + 1)*uxx(i) + x
        L2 = (y**2 + 1)*uyy(i) + y*uy(i) + y
        L3 = uxy(i)
        fval(i) = (L1 + L2 + L3)*u(i) + hold

        L1 = (x**2 + 1)*uxx(n+i) + x
        L2 = (y**2 + 1)*uyy(n+i) + y*uy(n+i) + y
        L3 = uxy(n+i)
        fval(n+i) = (L1 + L2 + L3)*u(n+i) + hold
```

```
      end do


      return
      end
```

Instead the boundary conditions for y receive their own functions, similar to how the boundary conditions are defined for BACOLI. Since (4.6) has a cross derivative term the user must provide two things: byu, the exact solution value at the boundary, and byux, the x partial derivative value at the boundary. The left boundary condition for y is

```
      subroutine bndya(t, x, byu, byux)
c-----------------------------------------------------------------------
c purpose:
c       the subroutine is used to define the boundary conditions at the
c       first y boundary condition, y = yval(1).
c       The user must also provide the x partial derivative if the
c       system contains the cross derivative, uxy. Otherwise byux can
c       safely be ignored.
c
c-----------------------------------------------------------------------
c subroutine parameters:
c input:
c
        double precision        t
c                               the current time coordinate.
c
        double precision        x
c                               the current x coordinate.
c
c
c output:
        double precision        byu
c                               byu is the exact solution to the system
c                               at (x,yval(1),t)
c
        double precision        byux
c                               byux is x partial derivative
```

```
c                               at (x,yval(1),t). Can be ignored if
c                               ut is not dependant on uxy.
c-----------------------------------------------------------------------
        integer ymax
        parameter (ymax = 80)
        double precision a, b, w1, w2, z1, z2, l, gamma, eta
        common /tumor/ a, b, w1, w2, z1, z2, l, gamma, eta
        double precision xp(5), yp(5), yvals(ymax), e
        common /tumor/ xp, yp, yvals, e
        integer tumNum
        common /tumor/ tumNum
c-----------------------------------------------------------------------
        double precision        pi
c loop indices:
        integer                 i, n
c-----------------------------------------------------------------------
      n = npde/2
      pi = 3.1415926535
      byu = (e**(-t) + 1)*sin(pi*x)*sin(pi*yvals(1))
      byux = pi*((e**(-t) + 1))*sin(pi*yvals(1))*cos(pi*x)


c
      return
      end
```

and the right boundary condition for y is

```
      subroutine bndyb(t, x, npde, byu, byux)
c-----------------------------------------------------------------------
c purpose:
c       the subroutine is used to define the boundary conditions at the
c       second y boundary condition, y = yval(n).
c       The user must also provide the x partial derivative if the
c       system contains the cross derivative, uxy. Otherwise byux can
c       safely be ignored.
c
c-----------------------------------------------------------------------
```

```
c subroutine parameters:
c input:
c
        double precision        t
c                               the current time coordinate.
c
        double precision        x
c                               the current x coordinate.
c
        integer                 npde
c                               the number of pdes in the system.
c
c
c output:
        double precision        byu
c                               byu is the exact solution to the system
c                               at (x,yval(1),t)
c
        double precision        byux
c                               byux is x partial derivative
c                               at (x,yval(1),t). Can be ignored if
c                               ut is not dependant on uxy.
c-----------------------------------------------------------------------
        integer ymax
        parameter (ymax = 80)
        double precision a, b, w1, w2, z1, z2, l, gamma, eta
        common /tumor/ a, b, w1, w2, z1, z2, l, gamma, eta
        double precision xp(5), yp(5), yvals(ymax), e
        common /tumor/ xp, yp, yvals, e
        integer tumNum
        common /tumor/ tumNum
c-----------------------------------------------------------------------
c loop indices:
        integer                 i, n
c-----------------------------------------------------------------------
```

```
      n = npde/2
      pi = 3.1415926535
      byu = (e**(-t) + 1)*sin(pi*x)*sin(pi*yvals(n+2))
      byux = pi*((e**(-t) + 1))*sin(pi*yvals(n+2))*cos(pi*x)


c
      return
      end
```

Boundary conditions for x are defined the same way in both methods. The left x boundary condition is

```
      subroutine bndxa(t, u, ux, bval, npde)
c-----------------------------------------------------------------------
c purpose:
c       the subroutine is used to define the boundary conditions at the
c       left spatial end point x = xa.
c                           b(t, u, ux) = 0
c
c-----------------------------------------------------------------------
c subroutine parameters:
c input:
        integer                 npde
c                               the number of pdes in the system.
c
        double precision        t
c                               the current time coordinate.
c
        double precision        u(npde)
c                               u(1:npde) is the approximation of the
c                               solution at the point (t,xa).
c
        double precision        ux(npde)
c                               ux(1:npde) is the approximation of the
c                               spatial derivative of the solution at
c                               the point (t,xa).
```

```
c
c output:
        double precision        bval(npde)
c                               bval(1:npde) is the boundary condition
c                               at the left boundary point.
c-----------------------------------------------------------------------
        integer ymax
        parameter (ymax = 80)
        double precision a, b, w1, w2, z1, z2, l, gamma, eta
        common /tumor/ a, b, w1, w2, z1, z2, l, gamma, eta
        double precision xp(5), yp(5), yvals(ymax), e
        common /tumor/ xp, yp, yvals, e
        integer tumNum
        common /tumor/ tumNum
c-----------------------------------------------------------------------
c loop indices:
        integer                 i, n
c-----------------------------------------------------------------------
      n = npde/2
      pi = 3.1415926535
      do i = 1, n
         bval(i) = u(i) - (e**(-t) + 1)*sin(pi*a)*sin(pi*yvals(i+1))
         bval(n + i) = u(n+i)-(e**(-t) + 1)*sin(pi*a)*sin(pi*yvals(i+1))
      end do


c
      return
      end
```

and the right x boundary condition is

```
      subroutine bndxb(t, u, ux, bval, npde)
c-----------------------------------------------------------------------
c purpose:
c       the subroutine is used to define the boundary conditions at the
c       right spatial end point x = xb.
c                           b(t, u, ux) = 0
```

```
c
c-----------------------------------------------------------------------
c subroutine parameters:
c input:
        integer                 npde
c                               the number of pdes in the system.
c
        double precision        t
c                               the current time coordinate.
c
        double precision        u(npde)
c                               u(1:npde) is the approximation of the
c                               solution at the point (t,xb).
c
        double precision        ux(npde)
c                               ux(1:npde) is the approximation of the
c                               spatial derivative of the solution at
c                               the point (t,xb).
c
c output:
        double precision        bval(npde)
c                               bval(1:npde) is the boundary condition
c                               at the right boundary point.
c-----------------------------------------------------------------------
        integer ymax
        parameter (ymax = 80)
        double precision a, b, w1, w2, z1, z2, l, gamma, eta
        common /tumor/ a, b, w1, w2, z1, z2, l, gamma, eta
        double precision xp(5), yp(5), yvals(ymax), e
        common /tumor/ xp, yp, yvals, e
        integer tumNum
        common /tumor/ tumNum
c-----------------------------------------------------------------------
c loop indices:
        integer                 i, n
```

```
c-----------------------------------------------------------------------
      n = npde/2
      pi = 3.1415926535
      do i = 1, n
         bval(i) = u(i)-(e**(-t) + 1)*sin(pi*b)*sin(pi*yvals(i+1))
         bval(n + i) = u(n+i)-(e**(-t) + 1)*sin(pi*b)*sin(pi*yvals(i+1))

      end do
c
      return
      end
```

Initial conditions are also defined in the same manner when using either boundary condition style.

```
      subroutine uinit(x, u, npde)
c-----------------------------------------------------------------------
c purpose:
c       this subroutine is used to return the npde-vector of initial
c       conditions of the unknown function at the initial time t = t0
c       at the spatial coordinate x.
c
c-----------------------------------------------------------------------
c subroutine parameters:
c input:
        double precision        x
c                               the spatial coordinate.
c
        integer                 npde
c                               the number of pdes in the system.
c
c output:
        double precision        u(npde)
c                               u(1:npde) is vector of initial values of
c                               the unknown function at t = t0 and the
c                               given value of x.
c-----------------------------------------------------------------------
```

```
      double precision pi
      parameter       (pi = 3.14159265358979323846264d0)
c-----------------------------------------------------------------------
        integer ymax
        parameter (ymax = 80)
        double precision a, b, w1, w2, z1, z2, l, gamma, eta
        common /tumor/ a, b, w1, w2, z1, z2, l, gamma, eta
        double precision xp(5), yp(5), yvals(ymax), e
        common /tumor/ xp, yp, yvals, e
        integer tumNum
        common /tumor/ tumNum
c-----------------------------------------------------------------------
c loop indices:
        integer                 i
c-----------------------------------------------------------------------
c local variables
        double precision        temp
c-----------------------------------------------------------------------
c
c     assign u(1:npde) the initial values of u(t0,x).
c
      do i = 1, npde/2
         u(i) = 1/(1+e**( (yvals(i+1) + x)/(2*eta) ) )
         u(npde/2 + i) = 1/(1+e**( (yvals(i+1) + x)/(2*eta) ) )
      end do
c
      return
      end
```

All other usage is based on setting up and running BACOLI. Information on BACOLI can be found in [9].

A.3 Test Problem (4.3) in Scilab

To use the Scilab code, the user needs to define t as a vector representing the t domain of the problem, set start to be the left boundary point of the x domain, and fin to be the right boundary

point of the x domain. func(1) is the left boundary condition and func(n) is the right boundary condition. func(2) to func(n-1) must describe the PDE. The value for g defines for how many iterations the test will run.

```
//clear
//This section of code applies the finite difference
scheme on the spatial dimension
//Leaving odes to be solved by the ode solver.
clear
//x = [0:0.01:1]
//n = length(x)
t = [0:0.01:1]
eps = 0.01
pi = 3.14159265358979
e = 2.7182818284590452

start = 0
fin = 1
m = 20//Initial mesh, doubled each run (including first run)
a = start
b = fin

x(1) = a
x((m+1)/2 + 1) = (b-a)/2 + a
for i=3:m/2 + 1
    x(m/2 - i+3)= (b-a)/i + a
    x(m/2 + i - 1) = b - (b-a)/i
end
x(m+1) = b
n = length(x)

for g = 1:5

for z = 1:m
    xt(2*z - 1) = x(z)
    xt(2*z) =  (x(z) + x(z+1))/2
```

```
end
xt(2*m+1) = b
x = xt
m = 2*m
n = m + 1



for i = 1:n
    init(i) = (1/2) - (1/2)*tanh( (1/(4*eps))*(x(i) - 0.25))
end



function func=od(t, y)
    //y(1) = 1
func(1) = ((sech((1/(4*eps))*(-(t/8) - 0.25)))**2)/(64*eps)

h1 = x(2) - x(1)
h2 = x(3) - x(2)
h3 = x(4) - x(2)
h4 = x(5) - x(2)

c1 = 2*(h2*h3 + h2*h4 + h3*h4)/(h1*(h1**3 + h1**2*h2
+ h1**2*h3 + h1**2*h4 + h1*h2*h3 + h1*h2*h4 + h1*h3*h4 + h2*h3*h4))
c2 = 2/(h3*h4) + 2/(h2*h4) + 2/(h2*h3) - 2/(h1*h4) - 2/(h1*h3) - 2/(h1*h2)
c3 = 2*(-h1*h3 - h1*h4 + h3*h4)/(h2*(h1*h2**2 - h1*h2*h3
- h1*h2*h4 + h1*h3*h4 + h2**3 - h2**2*h3 - h2**2*h4 + h2*h3*h4))
c4 = 2*(h1*h2 + h1*h4 - h2*h4)/(h3*(h1*h2*h3 - h1*h2*h4
- h1*h3**2 + h1*h3*h4 + h2*h3**2 - h2*h3*h4 - h3**3 + h3**2*h4))
c5 = 2*(-h1*h2 - h1*h3 + h2*h3)/(h4*(h1*h2*h3 - h1*h2*h4
- h1*h3*h4 + h1*h4**2 + h2*h3*h4 - h2*h4**2 - h3*h4**2 + h4**3))
uxx = (c5*y(2+3) + c4*y(2+2) + c3*y(2+1) + c2*y(2) + c1*y(2-1))

c1 = -h2*h3*h4/(h1*(h1**3 + h1**2*h2 + h1**2*h3 + h1**2*h4
+ h1*h2*h3 + h1*h2*h4 + h1*h3*h4 + h2*h3*h4))
c2 = -1/h4 - 1/h3 - 1/h2 + 1/h1
```

```
c3 = h1*h3*h4/(h2*(h1*h2**2 - h1*h2*h3 - h1*h2*h4 + h1*h3*h4
+ h2**3 - h2**2*h3 - h2**2*h4 + h2*h3*h4))
c4 = -h1*h2*h4/(h3*(h1*h2*h3 - h1*h2*h4 - h1*h3**2 + h1*h3*h4
+ h2*h3**2 - h2*h3*h4 - h3**3 + h3**2*h4))
c5 = h1*h2*h3/(h4*(h1*h2*h3 - h1*h2*h4 - h1*h3*h4 + h1*h4**2
+ h2*h3*h4 - h2*h4**2 - h3*h4**2 + h4**3))
ux = (c5*y(2+3) + c4*y(2+2) + c3*y(2+1) + c2*y(2) + c1*y(2-1))

func(2) = 0.25*(eps*uxx -y(2)*ux)

for i = 3:(n-2)
    h1 = x(i) - x(i-2)
    h2 = x(i) - x(i-1)
    h3 = x(i+1) - x(i)
    h4 = x(i+2) - x(i)

    c1 = 2*(-h2*h3 - h2*h4 + h3*h4)/(h1*(h1**3 - h1**2*h2 + h1**2*h3
+ h1**2*h4 - h1*h2*h3 - h1*h2*h4 + h1*h3*h4 - h2*h3*h4))//a
    c2 = 2*(h1*h3 + h1*h4 - h3*h4)/(h2*(h1*h2**2 + h1*h2*h3
+ h1*h2*h4 + h1*h3*h4 - h2**3 - h2**2*h3 - h2**2*h4 - h2*h3*h4))
    c3 = 2/(h3*h4) - 2/(h2*h4) - 2/(h2*h3) - 2/(h1*h4) - 2/(h1*h3) + 2/(h1*h2)
    c4 = 2*(h1*h2 - h1*h4 - h2*h4)/(h3*(h1*h2*h3 - h1*h2*h4
+ h1*h3**2 - h1*h3*h4 + h2*h3**2 - h2*h3*h4 + h3**3 - h3**2*h4))
    c5 = 2*(-h1*h2 + h1*h3 + h2*h3)/(h4*(h1*h2*h3 - h1*h2*h4
+ h1*h3*h4 - h1*h4**2 + h2*h3*h4 - h2*h4**2 + h3*h4**2 - h4**3))//e
    uxx = (c5*y(i+2) + c4*y(i+1) + c3*y(i) + c2*y(i-1) + c1*y(i-2))

    c1 = h2*h3*h4/(h1*(h1**3 - h1**2*h2 + h1**2*h3
+ h1**2*h4 - h1*h2*h3 - h1*h2*h4 + h1*h3*h4 - h2*h3*h4))//a
    c2 = -h1*h3*h4/(h2*(h1*h2**2 + h1*h2*h3
+ h1*h2*h4 + h1*h3*h4 - h2**3 - h2**2*h3 - h2**2*h4 - h2*h3*h4))
    c3 = -1/h4 - 1/h3 + 1/h2 + 1/h1
    c4 = -h1*h2*h4/(h3*(h1*h2*h3 - h1*h2*h4 + h1*h3**2
- h1*h3*h4 + h2*h3**2 - h2*h3*h4 + h3**3 - h3**2*h4))
    c5 = h1*h2*h3/(h4*(h1*h2*h3 - h1*h2*h4 + h1*h3*h4
```

```
- h1*h4**2 + h2*h3*h4 - h2*h4**2 + h3*h4**2 - h4**3))//e
    ux = (c5*y(i+2) + c4*y(i+1) + c3*y(i) + c2*y(i-1) + c1*y(i-2))

    func(i) = 0.25*(eps*uxx -y(i)*ux)
end

h1 = x(n-1) - x(n-4)
h2 = x(n-1) - x(n-3)
h3 = x(n-1) - x(n-2)
h4 = x(n) - x(n-1)

c1 = 2*(h2*h3 - h2*h4 - h3*h4)/(h1*(h1**3 - h1**2*h2 - h1**2*h3
+ h1**2*h4 + h1*h2*h3 - h1*h2*h4 - h1*h3*h4 + h2*h3*h4))
c2 = 2*(-h1*h3 + h1*h4 + h3*h4)/(h2*(h1*h2**2 - h1*h2*h3
+ h1*h2*h4 - h1*h3*h4 - h2**3 + h2**2*h3 - h2**2*h4 + h2*h3*h4))
c3 = 2*(h1*h2 - h1*h4 - h2*h4)/(h3*(h1*h2*h3 + h1*h2*h4
- h1*h3**2 - h1*h3*h4 - h2*h3**2 - h2*h3*h4 + h3**3 + h3**2*h4))
c4 = -2/(h3*h4) - 2/(h2*h4) + 2/(h2*h3) - 2/(h1*h4) + 2/(h1*h3) + 2/(h1*h2)
c5 = 2*(h1*h2 + h1*h3 + h2*h3)/(h4*(h1*h2*h3 + h1*h2*h4
+ h1*h3*h4 + h1*h4**2 + h2*h3*h4 + h2*h4**2 + h3*h4**2 + h4**3))
uxx = (c5*y(n) + c4*y(n-1) + c3*y(n-2) + c2*y(n-3) + c1*y(n-4))

c1 = -h2*h3*h4/(h1*(h1**3 - h1**2*h2 - h1**2*h3
+ h1**2*h4 + h1*h2*h3 - h1*h2*h4 - h1*h3*h4 + h2*h3*h4))
c2 = h1*h3*h4/(h2*(h1*h2**2 - h1*h2*h3 + h1*h2*h4
- h1*h3*h4 - h2**3 + h2**2*h3 - h2**2*h4 + h2*h3*h4))
c3 = -h1*h2*h4/(h3*(h1*h2*h3 + h1*h2*h4 - h1*h3**2
- h1*h3*h4 - h2*h3**2 - h2*h3*h4 + h3**3 + h3**2*h4))
c4 = -1/h4 + 1/h3 + 1/h2 + 1/h1
c5 = h1*h2*h3/(h4*(h1*h2*h3 + h1*h2*h4 + h1*h3*h4
+ h1*h4**2 + h2*h3*h4 + h2*h4**2 + h3*h4**2 + h4**3))
ux = (c5*y(n) + c4*y(n-1) + c3*y(n-2) + c2*y(n-3) + c1*y(n-4))

func(n-1) = 0.25*(eps*uxx -y(n-1)*ux)
```

```
func(n) = ((sech((1/(4*eps))*(-(t/8) + 0.75)))**2)/(64*eps)
endfunction

function func=second(t, y)
    func(1) = ((sech((1/(4*eps))*(-(t/8) - 0.25)))**2)/(64*eps)

    for i = 2:n-1
        h1 = x(i) - x(i-1)
        h2 = x(i+1) - x(i)

        c1 = 2/(h1*(h1 + h2))
        c2 = 2/(h2*(h1 + h2))
        uxx = c2*y(i+1) - (c1 + c2)*y(i) + c1*y(i-1)

        c1 = h2/(h1*(h1 + h2))
        c2 = h1/(h2*(h1+h2))
        ux = c2*y(i+1) + (c1 - c2)*y(i) - c1*y(i-1)

        func(i) = 0.25*(eps*uxx -y(i)*ux)
    end

    func(n) = ((sech((1/(4*eps))*(-(t/8) + 0.75)))**2)/(64*eps)
endfunction

[f] = ode(init,0,t, 10e-8, 10e-8, od)
[f2] = ode(init,0,t, 10e-8, 10e-8, second)

for j = 1:length(x)

    for k = 1:length(t)
        sol(j,k) = (1/2) - (1/2)*tanh( (1/(4*eps))*(x(j) - t(k)/8 - 0.25))
        err(j,k) = abs(sol(j,k) - f(j,k))
        err2(j,k) = abs(sol(j,k) - f2(j,k))
        errest(j,k) = abs(f(j,k) - f2(j,k))
    end
```

```
end
maxerr(g) = max(err)
avgerr(g) = mean(err)

maxerr2(g) = max(err2)
avgerr2(g) = mean(err2)
avgest(g) = mean(errest)
maxest(g) = max(errest)

plot3d(x,t,f, alpha = 90, theta = 0)

end
for q = 1:(g-1)
    rattmax(q) = maxerr(q+1)/maxerr(q)
    rattavg(q) = avgerr(q+1)/avgerr(q)
    rattmax2(q) = maxerr2(q+1)/maxerr2(q)
    rattavg2(q) = avgerr2(q+1)/avgerr2(q)
end
compmax(g) = maxerr(g)/maxerr2(g)
compavg(g) = avgerr(g)/avgerr2(g)
//plot(t,f)
```

A.4 Test Problem (4.2) in Scilab

The user must define start as the left boundary point of the t domain, fin as the right boundary point of the t domain, lower as the left boundary condition, and upper as the right boundary condition. func(1) and func(n) do not need to be altered. func(2) to func(n-1) must define the BVODE. The BVODE must be entered in the form

$$u_t - f(t, u, u_{tt}) = 0.$$

```
// Define function 'simple' that specifies
// the system of nonlinear equations
clear
e = 2.718281828459
```

```
pi = 3.14159265359
//n = 21; //Set this to be how many mesh points you want
start = 0
fin = 2.5
lower = 1/(28-3*start*start); //Set this to be your lower bound.
upper = 1/(28-3*fin*fin); //Set this to be your upper bound.
m = 10//Initial mesh, doubled each run (including first run)
a = start
b = fin

x(1) = a
x((m+1)/2 + 1) = (b-a)/2 + a
for i=3:m/2 + 1
    x(m/2 - i+3)= (b-a)/i + a
    x(m/2 + i - 1) = b - (b-a)/i
end

x(m+1) = b



for k = 2 : 7

for z = 1:m
    xt(2*z - 1) = x(z)
    xt(2*z) =  (x(z) + x(z+1))/2
end
xt(2*m+1) = b
x = xt
m = 2*m
n = m + 1

for i = 1:(m)
    h(i) = x(i+1) - x(i)
end
```

```
function func=simple(y)

//Here it will be the scheme minus the equation, since it needs to be of form
//df/dt - f(t) = 0
h1 = x(2) - x(1)
h2 = x(3) - x(1)
h3 = x(4) - x(1)
h4 = x(5) - x(1)
c1 = -(1/h4 + 1/h3 + 1/h2 + 1/h1)
c2 = -h2*h3*h4/(h1*(h1**3 - h1**2*h2 - h1**2*h3
- h1**2*h4 + h1*h2*h3 + h1*h2*h4 + h1*h3*h4 - h2*h3*h4))
c3 = h1*h3*h4/(h2*(h1*h2**2 - h1*h2*h3 - h1*h2*h4
+ h1*h3*h4 - h2**3 + h2**2*h3 + h2**2*h4 - h2*h3*h4))
c4 = -h1*h2*h4/(h3*(h1*h2*h3 - h1*h2*h4 - h1*h3**2
+ h1*h3*h4 - h2*h3**2 + h2*h3*h4 + h3**3 - h3**2*h4))
c5 = h1*h2*h3/(h4*(h1*h2*h3 - h1*h2*h4 - h1*h3*h4
+ h1*h4**2 - h2*h3*h4 + h2*h4**2 + h3*h4**2 - h4**3))



func(1) = y(1) - lower;

//Second point in, dif scheme
h1 = x(2) - x(1)
h2 = x(3) - x(2)
h3 = x(4) - x(2)
h4 = x(5) - x(2)
c1 = -h2*h3*h4/(h1*(h1**3 + h1**2*h2 + h1**2*h3 + h1**2*h4
+ h1*h2*h3 + h1*h2*h4 + h1*h3*h4 + h2*h3*h4))
c2 = -1/h4 - 1/h3 - 1/h2 + 1/h1
c3 = h1*h3*h4/(h2*(h1*h2**2 - h1*h2*h3 - h1*h2*h4
+ h1*h3*h4 + h2**3 - h2**2*h3 - h2**2*h4 + h2*h3*h4))
c4 = -h1*h2*h4/(h3*(h1*h2*h3 - h1*h2*h4 - h1*h3**2
+ h1*h3*h4 + h2*h3**2 - h2*h3*h4 - h3**3 + h3**2*h4))
```

```
c5 = h1*h2*h3/(h4*(h1*h2*h3 - h1*h2*h4 - h1*h3*h4
+ h1*h4**2 + h2*h3*h4 - h2*h4**2 - h3*h4**2 + h4**3))

func(2) = (c5*y(2+3) + c4*y(2+2) + c3*y(2+1) + c2*y(2)
+ c1*y(2-1)) - 6*y(2)*y(2)*x(2)
//func(2) = y(2) - e**(-2*x(2))

//Middle zone, all centered
for i = 3:(n-2)
    h1 = x(i) - x(i-2)
    h2 = x(i) - x(i-1)
    h3 = x(i+1) - x(i)
    h4 = x(i+2) - x(i)
    c1 = h2*h3*h4/(h1*(h1**3 - h1**2*h2 + h1**2*h3 + h1**2*h4
 - h1*h2*h3 - h1*h2*h4 + h1*h3*h4 - h2*h3*h4))//a
    c2 = -h1*h3*h4/(h2*(h1*h2**2 + h1*h2*h3 + h1*h2*h4
+ h1*h3*h4 - h2**3 - h2**2*h3 - h2**2*h4 - h2*h3*h4))
    c3 = -1/h4 - 1/h3 + 1/h2 + 1/h1
    c4 = -h1*h2*h4/(h3*(h1*h2*h3 - h1*h2*h4 + h1*h3**2
- h1*h3*h4 + h2*h3**2 - h2*h3*h4 + h3**3 - h3**2*h4))
    c5 = h1*h2*h3/(h4*(h1*h2*h3 - h1*h2*h4 + h1*h3*h4
- h1*h4**2 + h2*h3*h4 - h2*h4**2 + h3*h4**2 - h4**3))//e
    func(i) = (c5*y(i+2) + c4*y(i+1) + c3*y(i) + c2*y(i-1)
+ c1*y(i-2)) - 6*y(i)*y(i)*x(i)
end

//Second last point, different scheme
h1 = x(n-1) - x(n-4)
h2 = x(n-1) - x(n-3)
h3 = x(n-1) - x(n-2)
h4 = x(n) - x(n-1)
c1 = -h2*h3*h4/(h1*(h1**3 - h1**2*h2 - h1**2*h3 + h1**2*h4
+ h1*h2*h3 - h1*h2*h4 - h1*h3*h4 + h2*h3*h4))
c2 = h1*h3*h4/(h2*(h1*h2**2 - h1*h2*h3 + h1*h2*h4
- h1*h3*h4 - h2**3 + h2**2*h3 - h2**2*h4 + h2*h3*h4))
```

```
c3 = -h1*h2*h4/(h3*(h1*h2*h3 + h1*h2*h4 - h1*h3**2
- h1*h3*h4 - h2*h3**2 - h2*h3*h4 + h3**3 + h3**2*h4))
c4 = -1/h4 + 1/h3 + 1/h2 + 1/h1
c5 = h1*h2*h3/(h4*(h1*h2*h3 + h1*h2*h4 + h1*h3*h4
+ h1*h4**2 + h2*h3*h4 + h2*h4**2 + h3*h4**2 + h4**3))

func(n-1) = (c5*y(n) + c4*y(n-1) + c3*y(n-2) + c2*y(n-3)
+ c1*y(n-4)) - 6*y(n-1)*y(n-1)*x(n-1)
//func(n-1) = y(n-1) - e**(-2*x(n-1))

h1 = x(n) - x(n-4)
h2 = x(n) - x(n-3)
h3 = x(n) - x(n-2)
h4 = x(n) - x(n-1)
c1 = h2*h3*h4/(h1*(h1**3 - h1**2*h2 - h1**2*h3 - h1**2*h4
+ h1*h2*h3 + h1*h2*h4 + h1*h3*h4 - h2*h3*h4))
c2 = -h1*h3*h4/(h2*(h1*h2**2 - h1*h2*h3 - h1*h2*h4
+ h1*h3*h4 - h2**3 + h2**2*h3 + h2**2*h4 - h2*h3*h4))
c3 = h1*h2*h4/(h3*(h1*h2*h3 - h1*h2*h4 - h1*h3**2
+ h1*h3*h4 - h2*h3**2 + h2*h3*h4 + h3**3 - h3**2*h4))
c4 = -h1*h2*h3/(h4*(h1*h2*h3 - h1*h2*h4 - h1*h3*h4
+ h1*h4**2 - h2*h3*h4 + h2*h4**2 + h3*h4**2 - h4**3))
c5 = 1/h4 + 1/h3 + 1/h2 + 1/h1

func(n) = y(n) - upper
endfunction

//Second order function
function func=second(y)
    func(1) = y(1) - lower

    for i = 2:n-1
        h1 = x(i) - x(i-1)
        h2 = x(i+1) - x(i)
        c1 = h2/(h1*(h1 + h2))
```

```
        c2 = h1/(h2*(h1 + h2))

        func(i) = c2*y(i+1) + (c1 - c2)*y(i) - c1*y(i-1) - 6*y(i)*y(i)*x(i)
    end

    func(n) = y(n) - upper
endfunction


// Set initial guesses for the solution y
//x0 = [0:h:1]

// Call Scilab function 'fsolve' to solve the nonlinear system
[s] = fsolve(x,simple)
[s2] = fsolve(x,second)
//[sol] = fsolve(x0,true)


for i = 1:(n)
    sol(i) = 1/(28-3*x(i)*x(i))
    err(i) = s(i)- sol(i);
    err2(i) = s2(i) - sol(i)
    errest(i) = abs(s(i) - s2(i))
end

maxerr(k-1)=max(abs(err))
avgerr(k-1)=mean(abs(err))

maxerr2(k-1)=max(abs(err2))
avgerr2(k-1)=mean(abs(err2))

compormax(k-1) = maxerr(k-1)/maxerr2(k-1)
comporavg(k-1) = avgerr(k-1)/avgerr2(k-1)

maxest(k-1) = max(errest)
```

```
avgest(k-1) = mean(errest)


end
for q = 1:5
    rattmax(q) = maxerr(q+1)/maxerr(q)
    rattavg(q) = avgerr(q+1)/avgerr(q)
    rattmax2(q) = maxerr2(q+1)/maxerr2(q)
    rattavg2(q) = avgerr2(q+1)/avgerr2(q)
end
// Plot x vs. solution s
plot(x,s,x,s2)
//disp(x)
//disp(s)
```