Moving Mesh Methods for Solving Parabolic Partial Differential Equations

by

Robert Marlow

Submitted in accordance with the requirements for the degree of Doctor of Philosophy.

The University of Leeds

School of Computing

September 2010

Declarations

Some parts of the work presented in this thesis have been published in the following article:

Marlow, R., Hubbard, M. E. and Jimack, P. K. Moving mesh methods for solving parabolic partial differential equations. In *ICFD Conference on Numerical Methods for Fluid Dynamics*, 2010.

This article has also been submitted for publication to “Computers and Fluids”.

Abstract

In this thesis, we introduce and assess a new adaptive method for solving non-linear parabolic partial differential equations with fixed or moving boundaries, using a moving mesh with continuous finite elements. The evolution of the mesh within the interior of the spatial domain is based upon conserving the distribution of a chosen monitor function across the domain throughout time, where the initial distribution is based upon the given initial data. For the moving boundary cases, the mesh movement at the boundary is governed by a second monitor function. The method is applied with different monitor functions, to the semilinear heat equation in one space dimension, and the porous medium equation in one and two space dimensions. The effects of optimising initial data for chosen monitors will be considered - in these cases, maintaining the initial distribution amounts to equidistribution. A quantification of the effects of a mesh moving away from an equidistribution are considered here, also the effects of tangling, and then untangling a mesh and restarting.

Acknowledgements

I thank my supervisors Professor Peter Jimack and Dr Matthew Hubbard for their advice and enormous patience in the completion of this project. I would also like to thank all the many fellow students and academics for their help and support, particularly Dr Chris Goodyer and Dr Mark Walkley, in the School of Computing, at the University of Leeds. I would also like to thank Professor Mike Baines for his invaluable contributions to the many debates there have been on moving meshes, during this study. Lastly, but not least, I thank my dear wife Rosemary for checking the English in my thesis, and for many other things besides.

Contents

List of Figures

Chapter 1

Introduction

"Imagination is more important than knowledge." - Albert Einstein.

1.1 Background

This thesis extends a numerical technique developed by Baines, Hubbard and Jimack [4] to solve nonlinear parabolic partial differential equations (PDEs) with moving mesh methods. We will refer to the original algorithm as BHJ and the extension as BHJx.

Starting from the position of analysing physical problems, we are looking here at a *second* level of abstraction. That is to say, if the first level is to build a mathematical model of a physical problem that leads to a system of PDEs to solve, the second level is a numerical technique to solve the PDEs, but in some simplified, canonical form. For example, we might study the diffusion equation ($u_t = u_{xx}$) with no reference to a mathematical model, let alone any physical system.

This sets some guidelines, but also presents two main challenges for such a study. The guidelines amount to providing a general numerical technique, rather than solving a specific class of mathematical models. The first challenge is to identify an area where the current theory is worth extending. The second is to know the extension *could* be used by mathematical modellers, or by other theorists for *their* extensions of the theory - almost a

third level of abstraction there.

To address the first challenge, we note that moving mesh methods have been shown to have great potential in solving problems with moving fronts and boundaries, problems involving phenomena such as blow-up and problems in a wide range of applications for which non-stationary features need to be tracked in time (see, for example, [15] and references therein). However, there are many outstanding questions over accuracy and reliability remaining and new methods of evolving the mesh for a given problem are clearly worth considering.

On the second challenge, the question of applications, if there were no known applications of BHJ, this would be difficult. For some abstract theories, extensions have been created on more esoteric grounds of elegance and satisfaction [32], sometimes coupled with a rare insight that the extension will be useful in the physical world. However, we can definitely say that mathematical modellers, looking for specific key features in abstracted studies, have used the BHJ algorithm. This can be seen in two papers, both by Khassehkhan and Eberl. In the first [57], they have used BHJ specifically to provide a workable algorithm for a moving mesh study, including the need for a moving boundary. In the second paper [58], they have used BHJ as it provides a method of tracking steep interfaces. The boundary and interface requirements can be considered as examples of ”localized moving singularities” [31], so we can specifically say that this is a key feature mathematical modellers have used BHJ for. Since we are keeping this key feature, but extending the algorithm to other monitor functions, and also presenting the algorithm in a workable form, we claim this second challenge is met. Additionally, to go any further in ensuring that BHJx definitely could be used by mathematical modellers would violate the guidelines above, specifically that we wish to provide a general numerical technique, rather than solving a specific class of mathematical models.

Given the foregoing, it remains to prove the new algorithm actually works and assess its strengths and limitations. The thesis in its literal sense of providing new methods to solve

parabolic partial differential equations will then be complete. This is addressed in the work that follows, which has essentially five components:-

- Finite elements. The basics of this fundamental discretization technique, mainly from the point of view of a static mesh. The reader may be familiar with this area, but we suggest that at least the notation section (1.2.2) is studied, as it applies to the rest of the thesis from that point.

- Adaptive techniques - moving meshes. All adaptive techniques are reviewed, but we focus especially on adaptivity by allowing mesh points to move.

- BHJ and BHJx. We discuss the evolution and context of the original algorithm, which used only the mass monitor, and how it has been extended to more general monitors, and how it can use a *second* monitor to define boundary movement.

- Implementation of specific monitor functions and optimal initial meshes. The new algorithm is assessed here, by using different monitor functions and different PDEs. The effect of optimising the initial conditions is considered - this then amounts to equidistribution. In the final chapter, where the arc-length monitor is studied, an attempt is made to strictly enforce equidistribution. The algorithm also presented some challenges for the algorithm in this last chapter, giving an opportunity to mould and stretch it to meet these challenges.

- Analysis, discussion and future work. Conclusions and possible extensions, potentially to solving hyperbolic PDEs.

1.2 Finite Elements

Finite elements are an approximation technique, where a PDE is solved in a “weak” sense [27, 85]. The domain under consideration is effectively dissected into smaller pieces, and

an approximate piecewise polynomial solution found on each of these pieces. This set of pieces is known as a grid (or mesh). Finite elements started originally as an engineering method, with analysis following later [11, 85].

1.2.1 Basics

The finite element technique is an approximation technique, where the dependent variable in a PDE is approximated by a function that is mathematically simple (only a piecewise linear or other polynomial form), on a finite set of elements, approximating the domain of the PDE. In terms of classical analysis, we could say we have moved from an *infinitesimal* (dx) to a *finite* element δx [27]. Consider for example, this second order elliptic PDE [40]:-

$$-\triangle u \Big|_{\Omega} = f; \quad u \Big|_{\partial\Omega} = 0, \tag{1.1}$$

where Ω is some simply-connected bounded region [3] in $\mathbb{R}^n$. If we look at the simplest example, which is 1D piecewise linear elements, then the domain is an interval on the real line, and we take elements to be equal sections of that real line, as shown in Figure 1.1.

Figure 1.1: Finite Elements - 1D grid

As with finite differences, we are approximating u by calculating its values at the $N-1$ points (or "nodes") $x = h, x = 2h, \ldots x = (N-1)h$. In this case however, we are also

specifying the form of the approximation between these points - in this example, a linear form, so our approximation, which we will call u_h, is actually a piecewise linear function - see Figure 1.2.

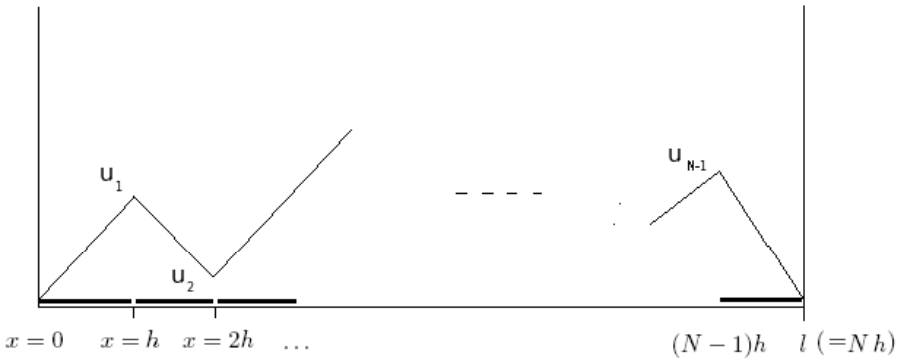

Figure 1.2: Finite elements - 1D approximating function

We can realise u_h as a linear combination of "trial functions" $w_1, w_2, \ldots w_{N-1}$ (see Figure 1.3), these being a basis of the *projection space* used to approximate u [27, 79, 85]:-

$$u_h = u_1 w_1 + u_2 w_2 + \cdots + u_{N-1} w_{N-1}. \tag{1.2}$$

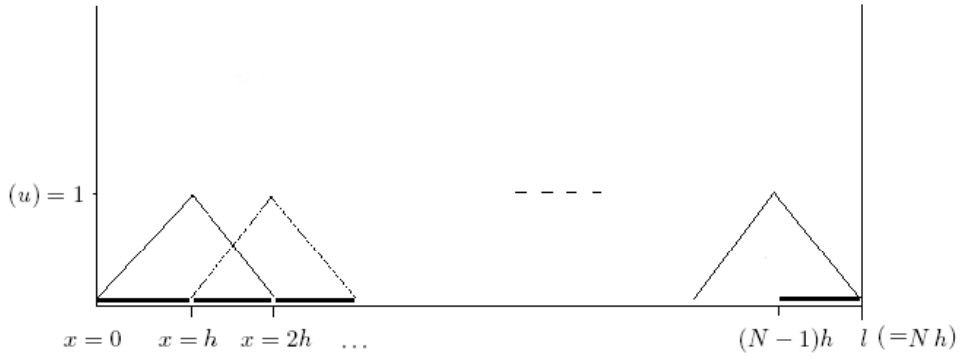

Figure 1.3: Finite elements - 1D standard trial functions

To calculate the value of u_h at the nodes, i.e., finding the u_i in equation (1.2), Green's first theorem [27] can be used to show that solving the PDE in (1.1) is equivalent to this statement [97]:-

$$\int_\Omega (\nabla u.\nabla v)\,dx = \int_\Omega f v\,dx, \text{ for all differentiable } v \text{ such that } v\Big|_{\partial\Omega} = 0. \qquad (1.3)$$

Now if u_h in equation (1.2) was actually a solution, we could write (1.3) as[1]:-

$$(\nabla u_h, \nabla v) = (f, v). \qquad (1.4)$$

That leads us to a method of approximation - we can *use* (1.4), with a set of *test functions* $\{v\}$ to find some $\{u_i\}$, and so solve the *weak* form of the PDE, i.e., approximately solve $-\triangle u = f$.

Suppose we use the trial functions themselves for the $\{v\}$. So if we substitute the trial functions $w_i, i = 1,2,\ldots N-1$ for v in (1.4), and expand u_h, we have:-

$$(\nabla(u_1 w_1 + u_2 w_2 + \cdots + u_{N-1} w_{N-1}), \nabla w_i) = (f, w_i), i = 1,2,\ldots N-1. \qquad (1.5)$$

Then writing $K_{i,j} = (\nabla w_i, \nabla w_j), (i = 1,2,\ldots N-1, j = 1,2,\ldots N-1)$ and $f_i = (f, w_i), i = 1,2,\ldots N-1$, as $(\nabla w_i, \nabla w_j) = (\nabla w_j, \nabla w_i)\ \forall i, j$, equation (1.5) can be written:-

$$\begin{aligned} u_1 K_{1,1} + u_2 K_{1,2} + \cdots + u_{N-1} K_{1,N-1} &= f_1 \\ u_1 K_{2,1} + u_2 K_{2,2} + \cdots + u_{N-1} K_{2,N-1} &= f_2 \\ &\vdots \\ u_1 K_{N-1,1} + u_2 K_{N-1,2} + \cdots + u_{N-1} K_{N-1,N-1} &= f_{N-1}, \end{aligned}$$

or equivalently:-

[1] Strang and Fix [85] also use this notation, though in some texts, if an *inner product* is being discussed, you may see $< \cdot, \cdot >$ instead. This text will always follow the $(\cdot, \cdot)$ notation.

$$K\underline{u} = \underline{f}, \tag{1.6}$$

which is known as the finite element equation, with K being referred to as the stiffness matrix [85], which we note is symmetric in this case. In practice, the simplicity of the trial functions (see Figure 1.3) means that K and $\underline{f}$ are fairly straightforward to calculate, and allows computational gains to be made (compared to a standard Gaussian elimination routine) in calculating $\underline{u}$, as K is sparse.

The addition of boundary conditions can be dealt with by noting the effects on the weak formulation, and hence the finite element equation, and then incorporating these into the finite element scheme. For example, a non-zero Dirichlet boundary condition (in the 1D case) of equation (1.1) at $x = 0$ can be allowed for by another trial function w_0 that has a fixed value u_0 at $x = 0$. Equation (1.2) can then be amended to become:-

$$\tilde{u}_h = u_0 w_0 + u_h \tag{1.7}$$

leading to relatively simple amendments to the finite element equation.

Strictly speaking, the trial (and here test) functions w_i are not differentiable, as required by (1.3). However, it is only necessary when calculating (v, w_i), for any integrable v, to integrate over the support of w_i, so the discontinuity in the derivative at the edge of the support is irrelevant.

The method of projecting u onto a finite approximation space existed before finite elements [85], but the basis then consisted of functions defined on the whole of the domain. This made it difficult - there then needs to be differentiable, or piecewise differentiable functions created to fit a domain with an awkward geometry (particularly an awkward boundary) when we have Dirichlet boundary conditions, and co-ordinate transformations [90] may not solve this problem. This is especially true if we have moving boundary problems (which form the bulk of the study in this thesis). The finite element

method, when using the form for this PDE in equation (1.3) solves this problem, as some of the elements will have an edge on the boundary.

When the span of the test functions is the same as that of the trial functions, as it is in our case, then this is known as the Galerkin method [27, 76], otherwise it is known as a Petrov-Galerkin method[2] [51]). Unless otherwise stated, only the Galerkin method is used in this thesis.

1.2.2 Notation

Unless otherwise stated, the symbol Ω always refers to some simply-connected bounded region [3] in $\mathbb{R}^n$, with $\partial\Omega$ referring to its boundary, and $\hat{\mathbf{n}}$ the outward pointing unit-length normal to $\partial\Omega$.

We will use the notation for the approximation (and discretization) for the dependent variables, as seen in Section 1.2.1, so that u_h always refers to the approximation of u. We reserve the letter h there, so that u_x, u_y, u_t refer to derivatives of u, and $u_i, u_j, \ldots$ (and $u_1, u_2, \ldots$) refer to the approximations of u at nodes $i, j, \ldots$ (and $1, 2, \ldots$). Rather than the N used in Section 1.2.1, in future, unless otherwise stated, N will always be the total number of nodes in a discretization, some of which may be boundary nodes, and so have forced, static values, when Dirichlet conditions are applied.

We will denote the trial functions (the basis for the solution space) by $w_1, w_2 \ldots w_N$. The above notation applies to other dependent variables, so for example, we may write the approximation of ϕ as $\phi_h = \Sigma_1^N \phi_i w_i$.

The equality we refer to as Green's lemma [54] is also known as the *divergence theorem*, which is that for a vector function, $\mathbf{F}$, and where such integrals exist:

$$\int_\Omega \nabla \cdot \mathbf{F} d\Omega = \int_{\partial\Omega} \mathbf{F} \cdot \hat{\mathbf{n}} dS. \tag{1.8}$$

[2] If a mesh is altered during a timestepping run, there can be different trial functions from the original mesh, these new trial functions being dependent on the PDE - effectively local solutions. This approach is known as multiscale [2]. The trial functions used in this thesis will always retain their essential form.

If this is combined with the product rule for a scalar function $\lambda(u)$ and a vector $\mathbf{F}$, that $\nabla\cdot(\lambda\mathbf{F}) = \nabla\lambda\cdot\mathbf{F} + \lambda\nabla\cdot\mathbf{F}$, then we get the "integration by parts" formula:

$$\int_\Omega \lambda\nabla\cdot\mathbf{F}d\Omega = \int_{\partial\Omega}\lambda\mathbf{F}\cdot\hat{\mathbf{n}}dS - \int_\Omega \mathbf{F}\cdot\nabla\lambda\, d\Omega. \tag{1.9}$$

When an algorithm is described, if we say, for example, that we will calculate some a from some b by using this weak form:-

$$\int_\Omega a\, w_i\, d\Omega = \int_\Omega \mathfrak{L}b\, w_i\, d\Omega,\ \ i = 1,2\ldots N,$$

for some linear operator $\mathfrak{L}$, then we mean that we are solving this by using the approximations $a_h = \Sigma_1^N a_i w_i$, $b_h = \Sigma_1^N b_i w_i$, and a_h is then found by solving the system of equations $K\mathbf{a} = \mathbf{c}\ (= P\mathbf{b})$ to find $\mathbf{a}$, where $K_{i,j} = (w_j, w_i)$, $\mathbf{a} = (a_1, a_2, \ldots a_N)^T$, $P_{i,j} = (\mathfrak{L}w_j, w_i)$ and $\mathbf{b} = (b_1, b_2, \ldots b_N)^T$, though we will give a more explicit description if this step is at all unclear.

1.2.3 Mesh Generation

As we saw in the analysis of the 1D case in Section 1.2.1, the practice of finite elements is to construct the stiffness matrix K, the "forcing" vector $\underline{f}$ and then solve the finite element equation $K\underline{u} = \underline{f}$. This practice becomes more complex however, when we solve problems in higher dimensions. We demonstrate this here for two dimensions, again looking at the Poisson equation.

In the first place, the concept of finite elements in 2D is the same as that in 1D, in that our approximation consists of simple functions built up on elements making up the computational domain. In the 2D case though, we might visualise this as a set of "plates" rather than single lines - see Figure 1.4.

As in the 1D case, it is possible to realise u_h as a linear combination of trial functions. To illustrate the piecewise linear trial functions, suppose our domain is the square $(0,l)\times$

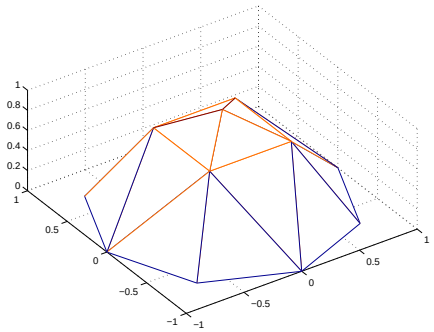

Figure 1.4: Finite element (2D plates) illustration

$(0, l)$ in $\mathbb{R}^2$ and our nodes and elements are arranged as in Figure 1.5.

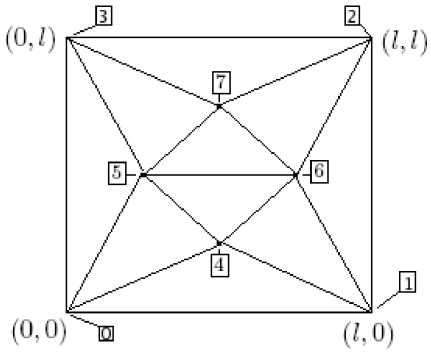

Figure 1.5: Finite element 2D grid

As with the 1D case, a trial function has a value of 1 at one node, and 0 at all others, and slopes linearly down to zero at neighbouring nodes. The trial function for node 6 (and $l = 1$) is shown in Figure 1.6.

Note though, that each of these trial functions can be realised as the sum of smaller "element" functions, for every element that has a node in common with the primary ("one-value") node of the trial function. For example, a trial function centred on node 5 in Figure 1.5 can be considered as the sum of five simpler functions on all the elements containing node 5. So when evaluating K, each $K_{i,j}$ can be built up from the element functions making up each trial function, therefore we can loop through the elements to find them, rather

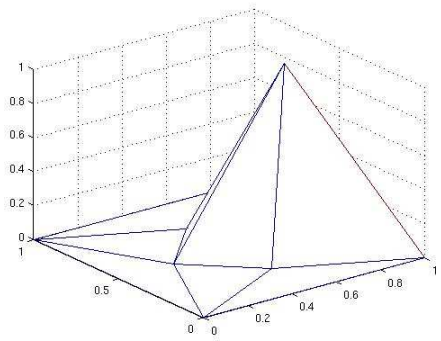

Figure 1.6: Finite elements - 2D trial function

than loop through the nodes, and so greatly simplify the calculation.

1.2.4 Connectivity

When modelling the PDE in equation (1.1) with the grid shown in Figure 1.5, the mathematics and the algorithm are essentially the same as described for the 1D example in Section 1.2.1, but there are some extra programming considerations regarding the 2D case.

Looking at nodes 4, 5, 6 and 7 in Figure 1.5 for example, it is not sufficient to just list the node co-ordinates. We also need to say that the triangles are formed by a line connecting nodes 5 and 6 as shown, since a line connecting nodes 4 and 7 is a possible alternative. We need then, a *connectivity* array, as well as defining the node positions. The convention chosen in this thesis is that we start, for each element, with the lower y co-ordinate, or lower left, then proceed anti-clockwise around the triangle. For example, the triangle on nodes 4, 5 and 6 in Figure 1.5 has a connectivity of "4 6 5". A possible input file describing this grid, when $l = 1$ is:-

Finite Elements. Square of 8 nodes.

0.0 1.0 0.0 1.01

8 10

0 0.000000000000000E+00 0.000000000000000E+00

0 0.100000000000000E+01 0.000000000000000E+00

0 0.100000000000000E+01 0.100000000000000E+01

0 0.000000000000000E+00 0.100000000000000E+01

1 0.500000000000000E+00 0.250000000000000E+00

1 0.250000000000000E+00 0.500000000000000E+00

1 0.750000000000000E+00 0.500000000000000E+00

1 0.500000000000000E+00 0.750000000000000E+00

0 4 5

0 1 4

1 6 4

0 5 3

4 6 5

1 2 6

5 7 3

5 6 7

6 2 7

7 2 3

22End of file.

The description of this is:-

Line 1: Title.

Line 2: Co-ordinates of Bottom left, Bottom right, Top left, Top right of Bounding Box.

This is for a Bounds check in the data.

Line 3: No. of nodes and No. of elements.

Line 4-11: Node flag (see below) and node co-ordinates.

Line 12-21: Element connectivity array, starting with “element O”.

Line 22: Integrity check - should be 22 lines.

The node flag is zero for a Dirichlet boundary node, and 1 for an internal node. The numbering direction for the nodes (in this example) is anti-clockwise for bounding nodes, starting near or on the origin, and then left to right, down to up. For the elements, it is left to right, then down to up. Neither of these numbering conventions are important for the algorithms in this thesis.

1.2.5 Element types and orders

In the 2D case, we have discussed the simplest form of element - piecewise linear on triangles. Higher order than linear can be used - see Section 1.3 below on "Adaptivity". It is also possible to use quadrilateral elements [34, 85] - these might be used to model a rectangular area more easily. As for meshes, the type described in the connectivity section, which is the only type used in this thesis, is actually referred to as *unstructured*, as we are allowing the element-node relationship to be anything we declare in the connectivity array. Alternatively, it is possible to use *structured elements* (repeated geometry), which need no connectivity array, and have a lot less cost than unstructured elements. For example, for rectangles, this could be just a uniform grid, for triangles, a grid where all the triangles might be bottom-left to top-right, so that in array/matrix terms, it is formulaic which elements are adjacent, and which nodes make up each element.
In practice, structured and unstructured can be mixed as appropriate, so for example, a smooth part of an aerofoil would be best modelled with structured elements [90].

1.2.6 Time-dependent semi-discretization, the Method of Lines

If our PDE has a time derivative, it is possible to simply add time as another dimension, so 2D triangles become tetrahedra for example. The problem with this approach is that there can then be no known boundary conditions for u at (some) $t = T$, so there is an open boundary, particularly with the parabolic PDEs considered in this thesis. For that reason, a timestepping approach is used in this study, with the timestep being the same

for all nodes. So what we are actually doing is discretizing only in space, or we could say *semi-discretizing* [14]. Looking at equation (1.2) for example, if we now have a PDE with a time derivative in it, so that the u_i become functions of time, the discretization can lead to solving a set of simultaneous *ODEs*[3] for the u_i. This solution technique is known as the *Method of Lines* [10, 14, 98]. This thesis will be using a Method of Lines approach, though not always in a way a standard ODE solver would use.

1.2.7 Comparison with Finite Differences

Finite difference schemes tend to have very low cost (CPU) compared to finite elements, they can actually be one third of the FE cost, because they do not have a mesh. They lose out though, with complex geometries, where for example, it is difficult to accurately model an awkward perimeter, and then if the PDE was 4th order say, the scheme is difficult to implement at the spatial boundary.

1.3 Adaptivity

Adaptivity is the process of changing a mesh during the course of a computation in response to changes in the dependent variable (or its approximation), to achieve greater accuracy and/or greater efficiency (and possibly also to improve stability) of the numerical scheme. As a simple example, if we were modelling pressure, and the pressure was changing rapidly (in time) in one area of the grid, we might want to alter the mesh around that area, possibly by adding more elements there, or re-distributing existing elements towards that region. Another use of adaptivity would be to cope with phase-change problems (liquid/gas etc), as studied by Burman [21] or McCarthy [69], where there could be an abrupt change in density in the region of the phase-change, and this needs to be captured with greater resolution than elsewhere.

[3] Some ODE solving methods allow a variable timestep, and can even have a temporary negative timestep to achieve convergence, but this timestep is still the same for all nodes.

The focus of this thesis is applying adaptivity with finite elements, but we should point out it can also be used with finite volumes and finite differences. For the former, an adaptive method is explained in Thomas and Lombard [89], and for the latter, there is a study of black holes with adaptive finite differences in Plewa [78].

We distinguish here between adapting a grid for the representation of the appropriate PDE solution, and the multigrid method [12, 38] where a coarser grid is temporarily used to speed up convergence.

1.3.1 h, p and r adaptivity

In finite element adaptivity, there are three main methods of refining a mesh:-

h-refinement - adding extra nodes.

p-refinement - increasing the order of the approximating polynomials.

r-refinement - changing position of existing nodes.

The focus of this thesis is r-refinement (the "r" is for re-distribution) in 1D and 2D, so this method is discussed in detail in Section 1.5. We conclude this section with a short review of h, p and r types of adaptivity. This review focuses on methods in 1D and 2D only, and we mainly consider schemes that have *conformity*, which we define generally here as the approximation space being a subspace of C^0. In 1D, for example, this would imply values agreeing at common nodes of adjacent segments [70].

It is possible to adapt nodes statically, i.e,. just adapt the mesh at each timestep, or dynamically, so that the mesh parameters (No. of nodes, positions, etc) are actually functions of time. This distinction is fully explained in Section 1.5, but for the purposes of this review, it is enough to mention that it is not necessary to just move from one point in time to the next in timestepping methods. In the *multistep* method [29, 53], a *history* of previous timesteps is used. This method can provide more accuracy, though it does add to the complexity of the problem, and may require any interpolation *between* nodes to be of consistent order [9].

h-refinement is named thus as h is commonly used for the spatial increment δx, which is reduced in selected areas of the grid, so correspondingly, the number of elements increases [62, 73]. In 1D, conformity is simple to enforce, if we define the elements as lines joining nodes. The situation is more complicated in 2D (or higher) though, where the way in which an element (such as a triangle) is sub-divided can determine whether conformity is maintained. In Figure 1.7 for example, we can see two ways in which a grid consisting of two triangles can be sub-divided. In the first sub-division, shown in the middle grid, we can see (at 'X'), there is a "hanging node": an internal node that is geometrically part of an element, yet that element (triangle T) does not have that node in its grid definition. The result is that the node value at X may not be the average of node values at A and B, so the approximation does not have conformity and will be inconsistent. The grid on the far right in Figure 1.7 shows an alternative sub-division which is conforming. Indeed, one way to achieve conformity is to avoid hanging nodes, and two different methods of achieving this are detailed in Bansch [7] and Speares and Berzins [84]. An alternative approach is to allow hanging nodes, but to constrain the solution values at these nodes [56] or have a choice of basis function at and near to those nodes which impose C^0 [87].

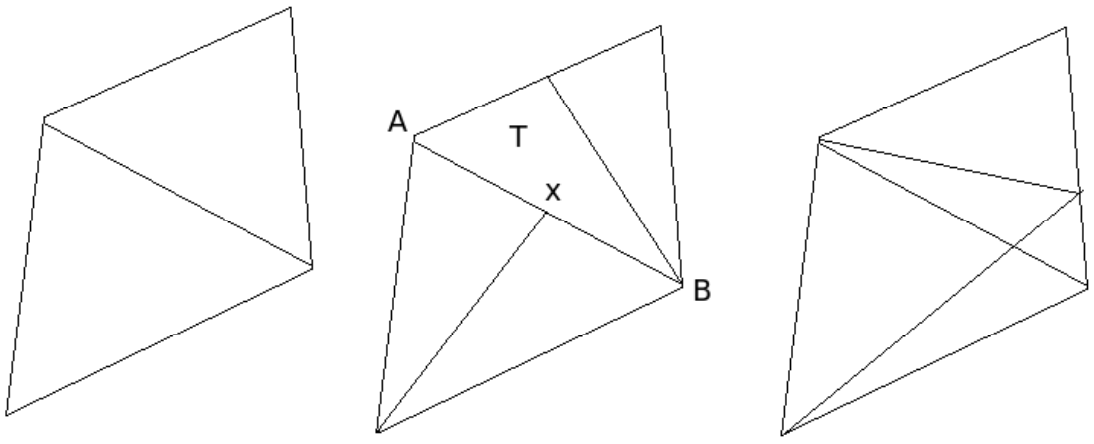

Figure 1.7: Conformity illustration

The advantages of h-refinement are that there will be no mesh "tangling" (elements overlapping) and a high accuracy can be achieved in a particular area. The disadvantages are

that it can be more difficult to use multistep, conformity issues as mentioned, and there can be a high CPU cost.

p-refinement is where higher-order (than linear) trial functions are used on the elements (hence p for polynomials). Conformity is harder to achieve here, especially in 2D and 3D, though it is certainly possible [23]. A recent popular alternative is to use a DG (Discontinuous Galerkin) method where the discontinuities are handled through the introduction of a "penalty" to a minimising integral (in the variational setting), hence the difference in a node value across elements is incorporated into a "flux integral", and this is added as a penalty function, whose minimisation is sought [82, 96]. As with h-refinement, there will be more CPU as a result of p-refinement, but better accuracy can be achieved. The advantages of p-refinement are that there will be no mesh "tangling"(elements overlapping) and a high accuracy can be achieved in a particular area, provided the solution has sufficient regularity in that area. There is an algorithmic simplicity in this method, since in this method, and only in this method, the number AND position of the nodes don't change in time. However, multistepping can still have problems, as interpolation between nodes may not be of consistent order. Comparing this method with h-refinement, in an area of smoothly oscillating values (see Figure 1.8 for example), a polynomial could be used for accuracy, but if the oscillations have low regularity (are only piecewise differentiable say, as in Figure 1.9), then it would actually be more efficient to add more points, in which case p-refinement would then be worse compared to h-refinement [80].

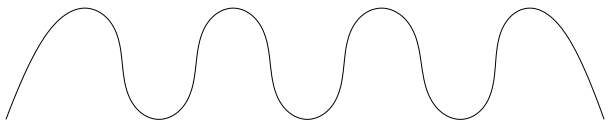

Figure 1.8: p-refinement illustration - smooth oscillations

r-refinement is defined as changing the position of existing nodes. The advantages of this method are simple algorithm management - as the method has a constant number of

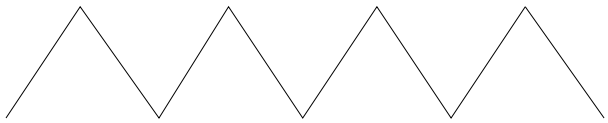

Figure 1.9: p-refinement illustration - irregular oscillations

points and topology, the matrices have a constant sparsity structure. So once a mesh and a program have been set up, there is no management of extra nodes that has to be done. This can result in lower CPU cost, compared with h and p methods. It is possible with h-refinement, for example, to achieve the same accuracy in an area of "high activity" by refining nodes there, and then, if that activity desists, losing said nodes ("derefinement"). This clearly involves extra management and CPU, and is essentially losing the advantage of the *continuity* of the r-refinement method. The disadvantages of r-refinement are that it can be difficult to multistep, as the nodes have changed position. In fact, Davis [29] did actually look at multistepping with r-refinement, but found it was unnecessary, i.e., only the previous timestep was needed. It can also be limited in its accuracy - for some cases, there may not be enough nodes to achieve a given error threshold.

Further advantages of r-refinement can be found in Budd, Huang and Russell [15].

There is actually a form of mesh-adaptivity that can be considered neither h, p or r, and that is *edge-swapping*. Figure 1.10 shows a 2D example of this - no nodes are added or moved, and the interpolation order can be the same, but the edge swap can improve some error measure, or some function (equi)distribution per element. More details can be found in Piggott, Farrell, Wilson, Gorman and Pain [77].

1.4 Errors, Error Estimates and Spaces

There is an inherent error in the finite element method, due to it being a finite dimensional approximation [25,85]. But in 2D cases and higher, in the case of a non-polygonal domain

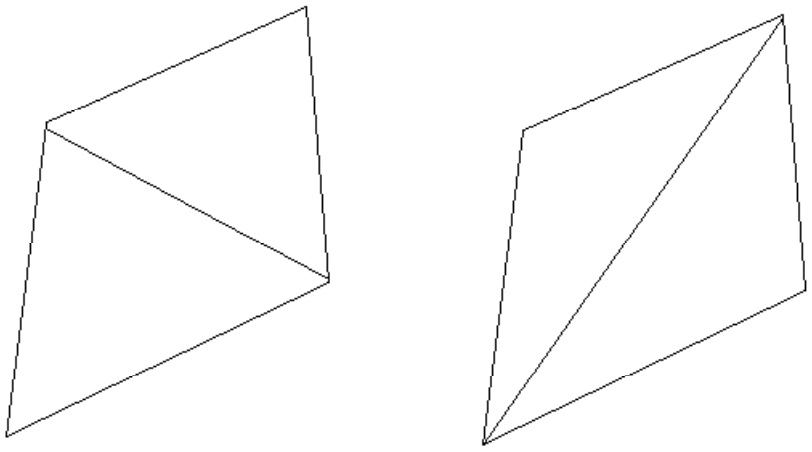

Figure 1.10: Edge-swapping example

shape, or rather its perimeter, there is also typically an error due to discretization of the boundary [79]. There could also be an error in mathematically modelling, though all the studies in this paper are of abstracted and simplified PDEs. The only other error is the inherent rounding error due to using computational techniques - this needs to be considered if we found, for example, that the difference in values of u_h at successive timesteps were close to machine precision [68]. In the studies of the porous medium equation in this thesis then, both L^2 and boundary errors will be given, as the analytic solution is known.

The minimum needed for an estimate is a lower AND an upper bound, and *Error Control* then means checking those bounds are not exceeded, using a suitable norm (Davis [29] uses L^2 for example). This can be done by element - analysing u_h on an element e actually gives an “error problem” for that element, we then need to solve another equation to find η_ε^h, this being the error from the true solution. This can itself be estimated, for example, by using the Bankweiser method [39]. There will, of course, be some CPU cost for this estimate. As another example of Error Control, in the porous medium equation studied in this thesis for example, as we know the solution is generally dissipative, a warning is

issued, or the program is halted, if the maximum value of $|u|$ at any time exceeds twice the value of the initial maximum value of $|u|$, as that definitely indicates a solution error, as we know the analytic solution.

The space used in approximation in this thesis is simply piecewise polynomial space, though there can be different bases, depending on our refinement type. For the solution, it depends on the application - it may be only L^2, up to C^∞, and in between these, it could be the Sobolev space [1] $W^{s,2}$, for some integer $s \geq 2$. For the second order PDEs we typically encounter, $W^{2,2}$ is really the bare minimum (we could take $W^{2,1}$, but that causes difficulties, and physically can amount to an infinite energy). We will not be exploiting the features of these spaces here though.

1.5 Moving Meshes

In r-refinement, a fixed number of mesh points are moved, but keeping the same mesh topology, hence the synonymous term "Moving Mesh". There are two methods of moving these points, *location-based* and *velocity-based*. In the former, a method is found to directly control the mesh points, in the latter, a method is used only to provide the mesh velocity, with the position being found by a timestepping scheme, such as Forward Euler, for example. These methods are explored in more detail in later sections, but this detail is enough for a brief overview of two of the papers in this area, which are themselves review papers. The emphasis of this overview has been a search for any moving mesh theory or guidelines that can be utilised or tested against in this thesis.

The first of these is Hawken, Gottlieb and Hansen [41]. Their review in 1991 is mostly focused on flow problems, using finite differences or finite elements. An extensive mathematical description is given of several methods, a few of them two-dimensional, followed by a comprehensive discussion of their relative merits. Although the authors have sometimes suggested improvements to these papers, there is no theory of moving meshes

developed here. This paper does not discuss h or p refinement.

The second review paper that we discuss is by Budd, Huang and Russell [15]. This review was published in 2009, and so has substantially more material than the first, but there is also a moving mesh theory developed here. This theory is almost exclusively applied to location-based methods. There are some 2D problems considered in this paper, but no 3D ones. Only r-refinement is considered in depth in this paper - h and p methods are discussed only briefly in the introduction. From this review, we can see if there is a "moving mesh strategy", it might be analytical if a location-based method is used, but is more likely to be empirical if using a velocity-based method. The method used in this thesis is a velocity-based method. Neither of these papers discusses elliptic problems.

1.5.1 Monitor functions - Concepts

Many moving mesh methods make use of a *monitor function*. We start with the idea of equidistribution in a mesh. This simply means re-arranging nodes so that some quantity is equally (as practically possible) distributed, other than just distance, area or volume. If we were looking at diffusion, this quantity might be mass. It doesn't have to be a physical quantity, if we want to cluster points near a "highspot" (see Figure 1.11), we might choose to equidistribute arc length.

The idea of the monitor function is to convert the intuitive but vague idea of "move the mesh to where activity is highest" into a solid mathematical statement, or at least something that can be numerically quantified. If we consider the porous medium equation for example [37, 59], which can model a gas "bubble" spreading in a porous medium [4], we might want to have more points where the mass density is highest. For a domain Ω, the monitor function would then simply be $m(u) = u$, and we would want to equidistribute $\int_\Omega m(u)dx$, so for example, in 1D, if our domain was $[0,1]$ which is to be sub-divided into $0 = x_0, x_1, x_2, \ldots x_N = 1$, then we would require that:-

$$\int_{x_i}^{x_{i+1}} m(u)\,dx = \frac{1}{N}\int_0^1 m(u)\,dx, \tag{1.10}$$

for $i = 0, 1, \ldots N-1$. As another example, referring to Figure 1.11, if we were trying to cluster our grid points near a "high spot", the diagram shows what would happen if we used arc length as the monitor function.

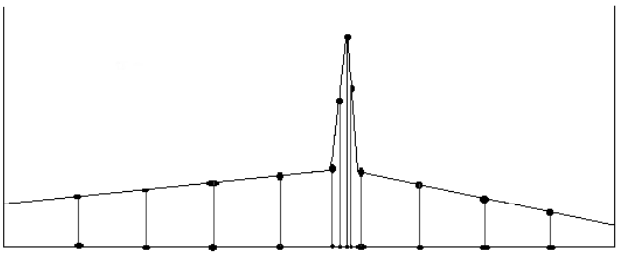

Figure 1.11: Adaptivity (arc length) illustration

When we look at equation (1.10), we can see a clue as how to actually achieve this equidistribution - at each timestep we could alter the node positions so that (1.10) is satisfied. This is called static re-gridding, but there are actually several ways of using these monitor functions, and then several ways of tying these ways to solving the PDE. These are considered in Section 1.5.2, where we look at how to apply monitor functions.
Following Budd, Huang and Russell [15], we will assume the monitor function is always non-negative. On types of monitor functions, Ren and Russell [81] claim there are three types of monitor function commonly used:-

- Arc length.
- Combination of gradient and curvature.
- Truncation error or solution residual.

Huang and Russell [49, 50] use an arc-length monitor. Budd, Huang and Russell [15] also use an arc-length monitor, and they show how it has modelled areas of high activity

successfully. This review paper also has details of other monitors used. An interesting example there is of a monitor function in a matrix form, which specifies the shape, size and orientation of the mesh elements.

We note two other monitors used:-

- A mass monitor ($m(u) = u$) is used by Baines, Hubbard and Jimack [4].
- A curvature monitor is used by Mackenzie and Robertson for a solution of a one-dimensional phase-field problem [65]. This monitor actually has a $sech(x)$ term in it. This is unusual as most monitor functions are only functions of u and its spatial derivatives, but one reason for having one as a function of x (and t) is to regulate behaviour near an interface - as Cai, Fleitas, Jiang and Liao [22] have done.

1.5.2 Monitor functions - Applying

Static regridding is discussed in Section 1.5.1 - this simply means resetting the node positions at each timestep, and a simple example of this is seen in Davis [29]. Even in this relatively simple 1D example though, we can see a problem emerging, that of "tangling" - referring to Figure 1.11, the node points can start to "over-distort", for example, cross over each other. Davis solves this problem by allowing the grid to not *exactly* equidistribute the monitor function (he calls this having a sub-optimal grid), but not actually collapsing, so the scheme is *stable*, i.e., it continues to its intended end-point in time.

We can see then that static regridding can be stable, however *dynamic* regridding, where we consider the node positions to be variables to be solved as well as solving the PDE [81], is generally more efficient, i.e., we can use fewer nodes *and* larger time steps to achieve the same error threshold [46]. Static regridding is also more difficult to apply in 2D problems. Referring to Figure 1.11, in actually calculating the new $x_0, x_1, \ldots$, we might, for example, start at the left-hand end, and slowly increment x until we get our "quantum" of the monitor integral in (1.10). In 2D though, there isn't really a natural direction to

"grow" our new distribution, and we could find it difficult to exactly allocate our N nodes successfully. For these reasons, we will only be considering dynamic regridding methods in this thesis. In fact, nearly all moving mesh methods use dynamic regridding, with static regridding being mostly being used by hp methods [15].

In all r-refinement methods, the underlying PDE and mesh equations can be solved by a Method of Lines approach, either singlestepping or multistepping (See Section 1.2.6).

The general aim of any r-refinement method is to maintain an "optimal geometry". Budd, Huang and Russell [15] define this, in practice, as equidistribution of a monitor function, but for the purpose of this thesis, we extend the definition to mean *maintaining the initial distribution* of a monitor function. If the initial mesh is optimised, meaning that the initial distribution is equidistributing the monitor function, then the two definitions concur, of course.

So any r-refinement method needs a mesh evolving mechanism that maintains this optimal geometry. The aim of this mechanism, and really the whole r-refinement method, is to have a better error for a given number of nodes, though we are also looking for efficiency and stability. For robustness of the method, the idea is to decrease the error for the same number (N) of points, so that the error only depends on N, (or dx equivalently).

We now discuss how to incorporate a moving mesh into solving a PDE. In a short review of 1D methods in 1992 by Ren and Russell [81], the view is taken that any equidistribution method amounts to a change in the co-ordinate system. However, we will continue with the distinction of location-based and velocity-based methods.

1.5.3 Location-based methods

All moving mesh methods can be considered as a mapping F from a computational space Ω_c to a physical space Ω_p, both subsets of $\mathbb{R}^d$ [98]. There is some theory developed in Budd, Huang and Russell [15] for F, for example "regularity" is defined as a measure of how much variation there is in the elements, so a uniform mesh is the most regular,

and this then leads to a mathematical definition in terms of the regularity (or smoothness) of F in a C^n sense. Since this theory is only applied to location-based methods and not velocity-based (which is the scope of this thesis), we will not discuss the theory further here, but just detail the main types of location-based methods, with examples.

The two main types of location-based methods are *optimal transport* and MMPDE (Moving Mesh PDE). The optimal transport method minimises $\int |F(\xi) - \xi|^2 d\xi$, so its aim is to be closest to a uniform mesh in a suitable norm, where ξ is the independent (spatial) variable in the computational domain (a symbol commonly used in the literature). This integral leads to a measure of deviation from the identity, so "smallest transport". F is actually written as the gradient of a mesh potential P, the mimimizing statement above then leads to an equation for P. More details can be found in Delzanno et al [30].

In the MMPDE method [16, 18, 24, 45–47, 49, 50, 98], the equidistribution requirement, or a variational version of it, is used to define a *second* PDE, that relates the equidistributed co-ordinate (ξ) to the fixed co-ordinate (x). This second PDE is the MMPDE. There was a review of MMPDE methods in 2001 by Huang and Russell [47], but as a simple example, we can take the 1D form of equidistribution in equation (1.10), and following White [95], re-write it for dynamic regridding thus:-

$$\int_0^{x(\xi,t)} m(u)\,dx = \xi\,\theta(t), \tag{1.11}$$

where $\theta(t) = \int_0^1 m(u)dx$, and $\xi \in [0,1]$. Following Huang and Russell [49, 50], we can differentiate (1.11) with respect to ξ:-

$$m(u)\frac{\partial x}{\partial \xi} = \theta(t). \tag{1.12}$$

And again:-

$$\frac{\partial}{\partial \xi}\left(m(u)\frac{\partial x}{\partial \xi}\right) = 0, \tag{1.13}$$

so giving our MMPDE, which will have to be solved in combination with solving the actual PDE. To illustrate this, suppose the monitor function was in the form $m(x)$. Then it may be possible to develop (1.11) to explicitly give $x = f(\xi)$, and recast the PDE in terms of the new independent spatial variable ξ. This is essentially the method studied in 1D by Budd and Piggott [16].

In the variational method of the MMPDE, the equidistribution requirement is realised as the minimising of an integral involving the monitor function (this may be done in a discrete form). This variational equation can then be used in the Euler-Lagrange form [6, 97] to form the MMPDE. This is the approach taken by Huang and Russell [45, 46] who study the problem in 2D. Zegeling and Kok [98] have studied it in 1D and 2D, looking at reaction-diffusion equations, with finite differences used for the numerical examples. We note that in Huang and Russell [45], the variational equation actually involves three monitors - as well as adaptivity, there is also mesh smoothness and "orthogonality". However, the authors only look at the MMPDE, and do not actually solve a PDE in this paper.

1.5.4 Velocity-based methods

Velocity-based methods use a Lagrangian (moving) co-ordinate system to directly provide a mesh velocity. A Lagrangian co-ordinate system is where the spatial co-ordinates are themselves functions of time, these functions being mappings from a conventional fixed co-ordinate system (this can be thought of as Ω_c in the location-based theory) to a moving one. The equidistribution question then becomes "How can this mapping be made to equidistribute the Lagrangian co-ordinates for the monitor function?". This question has been approached in three main ways: the Geometric Conservation Law method, the moving finite element method, and the Deformation Map method.

The Geometric Conservation Law (GCL) is akin to laws of mass and momentum conservation in fluid dynamics [52], but here we are saying it is *space* that must be conserved. This might seem self-evidently true, but if a finite element cell was to become

over-distorted, for example, the law could be violated. Mathematically, the Geometric Conservation Law is:-

$$\frac{d}{dt}\int_{A(t)} dx = \int_{\partial A(t)} \dot{\mathbf{x}} \cdot dS, \tag{1.14}$$

where $\dot{\mathbf{x}}$ is the mesh velocity and $A(t)$ is an arbitrary fixed cell in the finite element system. In 1979, Thomas and Lombard [89] used the GCL to simplify a finite difference calculation, so that it doesn't need "complicated averaging formulas", which some schemes not using the GCL do. In a similar vein, Étienne et al [35], and Farhat et al [36] have studied the GCL as an aid to quality of solutions, applied to fluid flow problems.

In studies by Cao, Huang and Russell [24] and Baines, Hubbard and Jimack [4] though, we see the GCL essentially being used as an algorithmic device to obtain mesh velocities, rather than guaranteeing mesh quality. In particular, we can use the GCL to eliminate the Jacobian that relates Ω_c to Ω_p and relate the mesh velocities directly to the monitor function as follows:-

Firstly, if we look at the moving mesh *method* in isolation from a PDE, we can study its features by letting the dependent variable u be some prescribed function, representing an exact physical solution. This is the way Huang, Ren and Russell [50] have studied the method, developing an MMPDE from the equidistribution principle, so although they have used the arc-length monitor in 1D ($m(\frac{\partial u}{\partial x}) = \sqrt{(}1 + (\frac{\partial u}{\partial x})^2$), they then study m as $m(x)$, not $m(u)$. Cao et al [24] have taken this abstraction a stage further by analysing forms of $m(x)$ with no actual reference to any PDE. Both these papers have proved their methods to a certain extent, though they left an open question of using the method to solve actual PDEs.

But staying with m as $m(x)$, if we differentiate equation (1.11) with respect to ξ, where m is now $m(x)$:-

$$m(x)\frac{\partial x}{\partial \xi} = \theta(t). \tag{1.15}$$

In 1D, the Jacobian is just $J = \frac{\partial x}{\partial \xi}$, so assuming (for the purposes of this analysis) a strictly positive $m(x)$, we can write (1.15) as:-

$$J = \frac{\theta(t)}{m(x,t)}. \tag{1.16}$$

In higher dimensions, this generalises as:-

$$J = \frac{\theta(t)}{m(\mathbf{x},t)}, \tag{1.17}$$

where $\theta(t)$ is now the monitor total normalised relative to the domain size [15]:-

$$\theta(t) = \frac{\int_{\Omega_p} m(\mathbf{x},t)d\Omega}{\int_{\Omega_c} d\xi}.$$

We now use the GCL to relate J to $\dot{\mathbf{x}}$, so that we can eliminate J and directly relate $\dot{\mathbf{x}}$ to the monitor function.

On the left-hand side of equation (1.14), if we change coordinates, so that corresponding to $A(t)$ in the physical space, there is a fixed cell A_c in the computational space Ω_c:-

$$\frac{\partial}{\partial t}\int_{A(t)} dx = \frac{\partial}{\partial t}\int_{A_c} J(\xi,t)d\xi = \int_{A_c} \frac{D}{Dt}(J(\xi,t))d\xi, \tag{1.18}$$

where $\frac{D}{Dt}$ is the total derivative $= \frac{\partial}{\partial t} + \dot{\mathbf{x}}\cdot\nabla$.

Applying Green's lemma to the right-hand side of (1.14), and then applying the change of co-ordinates:-

$$\int_{\partial A(t)} \dot{\mathbf{x}}\cdot dS = \int_{A(t)} \nabla\cdot\dot{\mathbf{x}}dx = \int_{A_c} \nabla\cdot\dot{\mathbf{x}}Jd\xi \tag{1.19}$$

As A_c is arbitrary, (1.18) and (1.19) together imply:-

$$\frac{D}{Dt}(J(\xi,t)) = \nabla \cdot \dot{\mathbf{x}} J. \tag{1.20}$$

Then assuming our monitor is non-degenerate, so that $\theta(t)$ in equation (1.17) is non-zero, we can write equation (1.20) as:-

$$\nabla \cdot \dot{\mathbf{x}} = J^{-1} \frac{DJ}{Dt}. \tag{1.21}$$

From equation (1.17):-

$$\frac{DJ}{Dt} = \frac{\dot{\theta}(t)}{m(\mathbf{x},t)} - \frac{\theta(t)}{m(\mathbf{x},t)^2} \frac{Dm(\mathbf{x},t)}{Dt} \Rightarrow$$

$$J^{-1} \frac{DJ}{Dt} = \frac{\dot{\theta}(t)}{\theta(t)} - \frac{1}{m(\mathbf{x},t)} \frac{Dm(\mathbf{x},t)}{Dt}. \tag{1.22}$$

So combining (1.21) and (1.22):-

$$\nabla \cdot \dot{\mathbf{x}} = \frac{\dot{\theta}(t)}{\theta(t)} - \frac{1}{m(\mathbf{x},t)} \frac{Dm(\mathbf{x},t)}{Dt} \Rightarrow$$

$$m(\mathbf{x},t) \nabla \cdot \dot{\mathbf{x}} + \frac{Dm(\mathbf{x},t)}{Dt} = m(\mathbf{x},t) \frac{\dot{\theta}(t)}{\theta(t)} \Rightarrow$$

$$m(\mathbf{x},t) \nabla \cdot \dot{\mathbf{x}} + \frac{\partial m(\mathbf{x},t)}{\partial t} + \dot{\mathbf{x}} \cdot \nabla m(\mathbf{x},t) = m(\mathbf{x},t) \frac{\dot{\theta}(t)}{\theta(t)} \Rightarrow$$

$$\nabla \cdot (m(\mathbf{x},t) \dot{\mathbf{x}}) + \frac{\partial m(\mathbf{x},t)}{\partial t} = m(\mathbf{x},t) \frac{\dot{\theta}(t)}{\theta(t)}. \tag{1.23}$$

Ren and Russell [81] have used (1.23) in 1D to solve for $\dot{\mathbf{x}}$ and $\dot{\theta}(t)$ simultaneously for a given monitor function. This same principle, of using (1.23) to get a mesh velocity from a monitor function has also been used in location-based methods. For example, Huang, Ren and Russell [50] develop an MMPDE method from (1.23), and then eliminate $\theta(t)$

from the system. In order to get uniqueness in solving for $\dot{\mathbf{x}}$ in 2D and higher dimensions, Cao et al [24] have developed a useful theoretical result to solve (1.23) (in the special case where $\dot{\theta}(t) = 0$), which is to realise $\dot{\mathbf{x}}$ as the gradient of a potential ϕ. An alternative to this method is a variational form to get $\dot{\mathbf{x}}$. It is only this latter form that is studied numerically in the Cao et al paper, and this confirms that the GCL is used here essentially as an algorithmic device, as some meshes did become heavily skewed, though they did not actually collapse.

The method of Baines, Hubbard and Jimack [4] which is described in Section 1.8, can also be considered a GCL method. This is actually a small step forward from the method of Cao et al [24], and in particular, has two key elements from that study - a method of using the monitor function to get mesh velocities, similar in form to equation (1.23) and realising the mesh velocity as the gradient of a potential ϕ, in a study of actual PDEs with moving boundaries. We will just mention here, that like Cao et al, using the GCL has not prevented tangling, so we again consider it an algorithmic device.

We conclude this section by discussing two other velocity-based methods. In the moving finite element (MFE) method [55, 71, 72, 93], the node positions are just added as extra variables to the finite element equation (1.6), but with connectivity remaining unaltered during refinement. Jimack [55] applies this method to the evolution equation (population growth), focusing on steady-state solutions. Note though, that the "new" finite element equation will no longer be a linear system. In the Deformation Map method, the equidistribution requirement is realised as a mapping between two domains in R^n, and the mesh velocities are constructed directly from this mapping [15]. A detailed study of this map has been made by Liao and Anderson [63] for a fixed boundary - the map is realised as n ODEs, n being the dimension of the domain (as in R^n).

1.5.5 Comparison of Location and Velocity-based methods

In the Ren and Russell 1992 review [81], both MMPDE and Lagrangian types can be seen, and this paper does have numerical examples given, with actual PDEs solved. Other comparisons tend to be more general and analytically based, the general consensus being that the velocity-based methods are easier to implement, but are more prone to tangling. The studies by Cao et al [24], and Baines, Hubbard and Jimack though, show that tangling is not a major problem. A definite advantage of velocity-based methods is that no distinction is needed for boundary velocities [15]. The location-based methods certainly lend themselves to more mathematical analysis. This does not appear to be an advantage in itself, but it does affect the strategy we use to decide which method to use - this might be analytical if a location method is used, but is more likely to be empirical if using velocity methods.

Budd, Huang and Russell [15] have stated that in velocity-based methods, the solution can move away from equidistributed solutions, though they have left an open question as to what difference this actually makes to solution errors, or other criteria (this point is actually addressed in this thesis, where we will attempt to force the maintaining of the initial distribution).

1.5.6 Mesh problems

1.5.6.1 Tangling

A tangled mesh is one whose elements are actually intersecting, i.e., part or all of any two elements are occupying the same space. In the 1D algorithm, we always assume the node indices run left to right, and in the 2D case, we have assumed the elements are numbered anti-clockwise. Therefore tangling is equivalent to element lengths or areas becoming less than or equal to zero, which is what is checked for in the algorithm. Results are generally unreliable following tangling, and we will normally consider a case completed

on this condition, with results only evaluated up to the point where the tangling occurs.

1.5.6.2 Blow-up

A possible problem with clustering mesh points near a region of high values (for example) of u (the dependent variable) is that if this region actually contains a singularity, an excessive proportion of the mesh could effectively disappear into the region. This is known as the *blow-up* problem. Budd and Williams [17] deal with this problem in a 2D case by defining a new monitor function that is the average of a regularized monitor function near the singularity and the old one away from it. This proved to be a stable method, and moved a "substantial fraction" of the mesh points away from the singularity.

1.5.7 Applications of Moving Mesh methods

We mention a few applications here, which illustrate the wide scope of moving mesh methods. A more comprehensive list can be found in Budd, Huang and Russell [15].

- Dorfi and Drury [33] use an MMPDE method to study Astrophysics problems in 1D, in particular, Sod's shock tube problem and a supernova explosion.
- Mackenzie and Robertson [65] use a velocity-based method for solving phase change (Stefan) problems in 1D.
- Tang and Tang [88] use an MMPDE method, in 1D and 2D, to study shock waves - this is in a study of hyperbolic conservation laws, using finite volumes.
- An application in meteorology is the Eady problem [15], which is used to model cyclones. Here, the Euler equations are modelled in 2D, the spatial co-ordinates being latitude and height, the dependent variables being air velocity, pressure and temperature. This is a location-based method, based on optimal transport ideas.

1.6 Parabolic PDEs: Examples

By a parabolic PDE, we mean one of the form:-

$$u_t = Lu, \tag{1.24}$$

where L is purely a second order spatial operator, and there is no second order time derivative term in this PDE. Two parabolic PDEs are studied in this thesis: the porous medium equation and the semilinear heat equation.

1.6.1 The Porous Medium Equation

The porous medium equation (PME) models gas flows in porous media, spreading liquids etc [37, 59, 92, 93].
In a simplified form, with initial values and Dirichlet boundary conditions, it is:-

$$\frac{\partial u}{\partial t} = \nabla \cdot (u^n \nabla u) \ (\mathbf{x} \in \Omega, t > 0), n \text{ being a positive integer; } u \Big|_{t=0} = u_0(\mathbf{x}); u \Big|_{\partial\Omega} = 0. \tag{1.25}$$

This has a known solution, for appropriate initial conditions, of the form [74]:-

$$u(r,t) = \begin{cases} \frac{1}{\lambda^d(t)}(1 - (\frac{r}{r_0\lambda(t)})^2)^{1/n}, & |r| \leq r_0\lambda(t), \\ 0, & |r| > r_0\lambda(t), \end{cases} \tag{1.26}$$

where d is the space dimension, r the usual radial co-ordinate and where:-

$$\lambda(t) = \left(\frac{t}{t_0}\right)^{\frac{1}{2+dn}} \text{ and } t_0 = \frac{{r_0}^2 n}{2(2+dn)}. \tag{1.27}$$

1.6.2 The Semilinear Heat Equation

The semilinear heat equation describes the temperature of a reacting medium, such as a burning gas [20]. With a fixed boundary and Dirichlet boundary conditions, it takes the form:-

$$u_t = \triangle u + u^p, \mathbf{x} \in \Omega, t > 0; u\Big|_{t=0} = u_0(\mathbf{x}); u\Big|_{\partial\Omega} = 0,\ p \text{ is a positive integer.} \quad (1.28)$$

No analytic solution of the semilinear heat equation is known to this author.

1.7 General factors in algorithm design

1.7.1 Closure

By *closure* we mean introducing another condition to a set of equations to make them uniquely solvable. There is nothing to say this has to be derived from physical principles, but at least, we would want a condition imposed that is not physically unrealistic. For example, Baines, Hubbard and Jimack [4] use a vorticity condition on the grid velocity, whereas Budd and Piggott [16], when looking specifically at the porous medium equation, impose a centre of mass condition on the grid *positions*.

1.7.2 Scale Invariance

For a *PDE* to be invariant actually means that the form of the PDE is unchanged under a set of transformations. Solutions that are unchanged under *some* of these transformations are referred to as self-similar solutions [66, 75]. Looking at the 1D porous medium equation ($u_t = (uu_x)_x$) for example, Budd and Piggott [16] consider four "continuous transformation groups", two translations and two scaling symmetries. Effectively these latter two are:-

$$t \to \lambda t, x \to \lambda^{\frac{1}{2}} x \tag{1.29}$$

$$t \to \lambda t, u \to \frac{u}{\lambda}. \tag{1.30}$$

The self-similar solutions are then $t^{\gamma} v(\frac{x}{t^{\beta}})$, where $2\beta - \gamma = 1$, for some new function v. By using mass conservation, we can deduce $\gamma = -\frac{1}{3}, \beta = \frac{1}{3}$, hence there are self-similar solutions of the form [16]:-

$$u(x,t) = t^{-\frac{1}{3}}(a - \frac{x^2}{t^{\frac{2}{3}}}), \tag{1.31}$$

for some constant a.

As a simpler example, if we look at the wave equation in 1D [97], a classical method of solving it is to change to new variables $\xi = x - ct, \eta = x + ct$ and find a very simple solution in terms of ξ and η. The scale invariance of this new co-ordinate system is expressed in classical terminology by saying that "along a characteristic", $\frac{x}{t}$ is constant.

Hence some PDEs have an underlying, unchanging physical or mathematical feature - a scale invariance, which we can exploit in designing the numerical techniques for their solution, for example, by finding a *scaling* or other transformation of co-ordinates and possibly the dependent variable, that gives us a new dependent variable that is constant or slower-moving with time, or a new time-related co-ordinate. This slower changing can lead to stability in solution methods [14, 16]. For example, in the blow-up problem [16,20], the value of u will become very large compared to the spatial co-ordinate x after a certain time. We can require that the numerical, i.e., discrete co-ordinates are aligned with this physical scaling. If using an MMPDE, this translates to requiring that the MMPDE be scale-invariant (under the same set of transformations that the PDE is), which itself leads to using a scale-invariant monitor function [16, 20].

The method used in this thesis is an extension of the Baines, Hubbard and Jimack algo-

rithm [4]. As the development of that algorithm was influenced by scale invariance, it will also be a consideration in this work. For example, when the semilinear heat equation in Chapter 3 is studied, the scale-invariant monitor functions employed by Budd, Huang and Russell [20] are used as a first approach.

1.7.3 Timestepping methods

The timestepping scheme chosen in this thesis is Forward Euler. The main disadvantage of this method is that it is explicit and so, for stability, a sufficiently small time step has to be chosen. It is also only first order, however in practice the stability restriction on the step size always dominates over accuracy because of the stiffness of the equations. To minimise run times, the timestep has only been lowered as far as it needs to be. Having set a timestep, as the spatial grid size dx has been halved, the timestep has been quartered. This is following typical practice in some finite difference methods, and can be seen to be an adequate system in the original BHJ study [4]. In the porous medium equation studies in this thesis, the timestep remains constant throughout the run. However, in the study of blow-up in Chapter 3, we allow the timestep to further reduce as the value of the dependent variable u becomes large.

1.8 The Method of Baines, Hubbard and Jimack

The BHJ method [4] is a velocity-based adaptive algorithm, as described in Section 1.5.4. The Geometric Conservation Law (GCL) has been used as an algorithmic device to derive the mesh velocity from the monitor function, but here, actual PDEs have been solved, and so we see the monitor function m as a function of u (in fact, $m(u) = u$), rather than the $m(x)$ in the abstracted study by Cao et al [24]. As with the Cao study, the equations for the mesh velocity $\dot{\mathbf{x}}$ have been solved uniquely by realising $\dot{\mathbf{x}}$ as the gradient of a potential ϕ. The finite element discretization is done with linear triangular elements. The PDEs

considered are all of parabolic type, as described in Section 1.6, so are of the form:-

$$u_t = Lu, \tag{1.32}$$

where L is purely a second order spatial operator, and there is no second order time derivative term.

The algorithm can generally be described as a Method of Lines approach, with a basic explicit one-step timestepping [61] method used to solve the resulting ODE system. However, it is only the mesh positions $\mathbf{x}$ and a "mass total" that are updated from the ODE system - the nodal values of u are then recovered from a "distributed conservation principle" [4], which can be visualised as "nodes carrying mass around".

We first define the mass total:-

$$\theta(t) = \int_{\Omega(t)} u d\Omega. \tag{1.33}$$

Then we calculate the distribution constants:-

$$c_i = \frac{1}{\theta(t)} \int_{\Omega(t)} w_i u d\Omega, \quad i = 1, 2 \ldots N, \tag{1.34}$$

where w_i are N piecewise linear basis functions, which form a partition of unity [4]. The "distributed conservation principle" is then that the c_i remain constant throughout the run. With this PDE set in a moving frame, so that Ω is changing in time, equation (1.34) provides a system to calculate u from the current values of θ and $\mathbf{x}$. But the same equation also provides the algorithmic device needed to calculate the mesh velocities, which we now detail. If we rewrite (1.34) as:-

$$c_i \theta(t) = \int_{\Omega(t)} w_i u d\Omega, \quad i = 1, 2 \ldots N. \tag{1.35}$$

As the c_i remain constant, applying Leibnitz's rule [60] (and using Green's lemma):-

$$\begin{aligned} c_i\dot{\theta}(t) &= \frac{d}{dt}\int_{\Omega(t)} w_i u d\Omega \\ &= \int_{\Omega(t)} \frac{\partial(w_i u)}{\partial t} d\Omega + \int_{\partial\Omega(t)} w_i u \dot{\mathbf{x}}\cdot\hat{\mathbf{n}} dS \\ &= \int_{\Omega(t)} w_i u_t d\Omega + \int_{\Omega(t)} u\frac{\partial w_i}{\partial t} d\Omega + \int_{\Omega(t)} \nabla\cdot(w_i u\dot{\mathbf{x}}) d\Omega, \quad i = 1,2\ldots N \end{aligned} \tag{1.36}$$

where $\hat{\mathbf{n}}$ is the outward pointing, unit-length normal at any point on the surface of Ω.

The w_i have been chosen to advect with velocity $\dot{\mathbf{x}}$ [4], therefore $\frac{\partial w_i}{\partial t} = -\dot{\mathbf{x}}\cdot\nabla w_i$, and hence:-

$$c_i\dot{\theta}(t) = \int_{\Omega(t)} w_i u_t d\Omega + \int_{\Omega(t)} u(-\dot{\mathbf{x}}\cdot\nabla w_i) d\Omega + \int_{\Omega(t)} \nabla\cdot(w_i u\dot{\mathbf{x}}) d\Omega, \quad i = 1,2\ldots N. \tag{1.37}$$

Expanding the last integral and simplifying:-

$$c_i\dot{\theta}(t) = \int_{\Omega(t)} w_i u_t d\Omega + \int_{\Omega(t)} w_i \nabla\cdot(u\dot{\mathbf{x}}) d\Omega, \quad i = 1,2\ldots N. \tag{1.38}$$

By using vorticity arguments [4], equation(s) (1.38) can be used to solve uniquely for $\dot{\mathbf{x}}$, if $\dot{\theta}$ and u are known and $\nabla\times\dot{\mathbf{x}}$ is specified. In fact, $\nabla\times\dot{\mathbf{x}}$ is specified as zero, so that $\dot{\mathbf{x}}$ can be written as $\dot{\mathbf{x}} = \nabla\phi$, for a velocity potential ϕ. The issue of finding $\dot{\theta}$ is addressed shortly, but first, writing $\dot{\mathbf{x}} = \nabla\phi$ in equation (1.38) and using the original PDE (1.32), we have:-

$$c_i\dot{\theta}(t) = \int_{\Omega(t)} w_i L u d\Omega + \int_{\Omega(t)} w_i \nabla\cdot(u\nabla\phi) d\Omega, \quad i = 1,2\ldots N. \tag{1.39}$$

In discrete terms, if we approximate ϕ by $\Sigma_{i=1}^{N}\phi_i w_i$, then we now have N equations in $N+1$ unknowns, because $\dot{\theta}$ is unknown as well as the ϕ_i. But we can add an equation for just $\dot{\theta}$ by developing equation (1.33) in a similar manner to above. By setting $w_i \equiv 1$ and

using the same arguments, we arrive at:-

$$\dot{\theta}(t) = \int_{\Omega(t)} Lu d\Omega + \int_{\Omega(t)} \nabla \cdot (u \nabla \phi) d\Omega. \tag{1.40}$$

However, these $N+1$ equations are actually singular as they stand, as equation (1.40) is effectively a sum of all the equations in (1.39). This is because the w_i are a partition of unity, as described earlier, and consequently the sum of the c_i is also unity, from the definition in (1.34). This can readily be rectified by eliminating one of the ϕ_i. Without loss of generality, we will actually set $\phi_1 = 0$ (and also only solve (1.39) for $i = 2, 3 \ldots N$), as it is only $\nabla\phi$ we are interested in [4]. Note that this is the only stage of the algorithm where the underlying PDE appears, as equation (1.34) is used as it stands to directly recover u from the current values of θ and $\mathbf{x}$.

Having found ϕ, we then calculate $\dot{\mathbf{x}}$ from the weighted form of $\dot{\mathbf{x}} = \nabla\phi$:-

$$\int_{\Omega(t)} w_i \dot{\mathbf{x}} d\Omega = \int_{\Omega(t)} w_i \nabla\phi d\Omega, \quad i = 1, 2 \ldots N. \tag{1.41}$$

Boundary conditions are only applied weakly, by disregarding the appropriate boundary integrals. For example, in equation (1.39), for the porous medium equation,

$$Lu = \nabla \cdot (u^n \nabla u).$$

So $\int_{\Omega(t)} w_i \nabla \cdot (u^n \nabla u) d\Omega$ will be calculated as:-

$$-\int_{\Omega(t)} u^n \nabla u \cdot \nabla w_i d\Omega,$$

thus weakly forcing $-\int_{\partial\Omega(t)} w_i u^n \nabla u \cdot \hat{\mathbf{n}} dS$ to be zero.

The complete algorithm is:

1. From initial values of u and $\mathbf{x}$, calculate the initial mass total θ and the monitor distribution constants c_i from (1.33) and (1.34).

2. Recover u from current values of θ and $\mathbf{x}$, using (1.35).

3. Solve equations (1.39) and (1.40) to find $\dot{\theta}$ and $\{\phi_i\}$.

4. Solve equation (1.41) to find $\dot{\mathbf{x}}$.

5. Update (with Forward Euler timestepping) $\mathbf{x}$ and θ, using $\dot{\mathbf{x}}$ and $\dot{\theta}$.

6. Return to step 2.

The above algorithm may be expressed more concisely as follows:-

Once only: Calculate θ, c_i

Loop:

$\theta * c_i \rightarrow u$.

$u \rightarrow (\phi, \dot{\theta}) \rightarrow \dot{\mathbf{x}}$.

Update θ and $\mathbf{x}$.

End Loop.

Chapter 2

BHJ - extended

We describe below, in Section 2.1, the main algorithm used in this thesis, in its general form. We refer to this as "BHJx". This extends the BHJ algorithm described in Section 1.8 to any monitor m of the form $m(u)$. It uses a differential form of the ALE to calculate $\dot{u}$ from $\dot{\mathbf{x}}$ [5], so that this can also be considered a Method of Lines approach (as described in Section 1.2.6). Here though, we are directly updating u and $\mathbf{x}$ at every timestep, from $\dot{u}$ and $\dot{\mathbf{x}}$. The finite element discretization is done with linear elements, triangular in the 2D case. The primary development history of the BHJx algorithm, leading up to the algorithm in 2.1, is described in Section 2.2 - numerous verification results are presented here.

2.1 The BHJx algorithm

Consider a general parabolic PDE, by which we mean one of the form:-

$$u_t = Lu, \tag{2.1}$$

on a time-dependent domain $\Omega(t)$, where L is purely a spatial operator of the second order. In particular, there is no second order time derivative term in this PDE. To make calculation of integral terms tractable, we further constrain L to be of the form $\nabla \cdot \mathbf{F} + G$,

where $\mathbf{F}$, G are operators on u, containing terms no higher than the first derivatives of u, for example $\mathbf{F} = \nabla u$, $G = u^p$ for the semilinear heat equation (discussed in Chapter 3).

2.1.1 ALE Formulation

In a Lagrangian system [4], with a moving co-ordinate $\mathbf{x}(t)$, u_t (the derivative with respect to the fixed system), actually has two components: the time derivative of u with respect to the moving frame, which we'll call $\dot{u}$, and a component from the movement of $\mathbf{x}$ itself, which is $-\nabla u \cdot \dot{\mathbf{x}}$, so equation (2.1) can be written in the form [5]:-

$$\dot{u} - \nabla u \cdot \dot{\mathbf{x}} = Lu, \tag{2.2}$$

which we can also write as:-

$$\dot{u} = Lu + \nabla u \cdot \dot{\mathbf{x}}. \tag{2.3}$$

This last form gives us a way of solving the PDE by timestepping methods, as (2.1) does, provided we have a means of defining $\dot{\mathbf{x}}$. The algorithm that we will employ can be generally described as calculating $\dot{u}, \dot{\mathbf{x}}$ from values of $u, \mathbf{x}$ at a given instant, then using $\dot{u}, \dot{\mathbf{x}}$ to update $u, \mathbf{x}$ at the next instant, and repeating the process forward in time.

In discrete terms, $\dot{\mathbf{x}}$ is the velocity of a *moving mesh* and u the values of the dependent variable at the nodes of the mesh, and we will actually calculate $\dot{u}$ for a given $\dot{\mathbf{x}}$ by using finite elements, and a weak differential form [5] of (2.3):-

$$\int_{\Omega(t)} w_i \dot{u} d\Omega = \int_{\Omega(t)} w_i (Lu + \nabla u \cdot \dot{\mathbf{x}}) d\Omega, \quad i = 1, 2 \ldots N, \tag{2.4}$$

where w_i are N piecewise linear basis functions, which form a partition of unity [4].

2.1.2 General Monitor Functions

The same distribution conservation ideas as developed by Baines, Hubbard and Jimack [4] to calculate $\dot{\mathbf{x}}$ from u are used, but we now extend it to more general monitor functions. So given a monitor function $m(u)$, we aim to maintain the distribution of the monitor total θ:-

$$\theta(t) = \int_{\Omega(t)} m(u) d\Omega. \tag{2.5}$$

This aim allows us to derive the mesh velocity $\dot{\mathbf{x}}$ from u, though we should stress that this is an algorithmic device for determining $\dot{\mathbf{x}}$, so we do not (generally) force the monitor distribution to be exactly maintained.

The first step in the algorithm is to derive distribution constants c_i. From initial values of u, we can calculate the value of $\theta(t)$ from (2.5). Using w_i as described in (2.4), we then calculate:-

$$c_i = \frac{1}{\theta(t)} \int_{\Omega(t)} w_i m(u) d\Omega, \quad i = 1, 2 \ldots N. \tag{2.6}$$

We now use these constants to calculate $\dot{\mathbf{x}}$ as follows. The weighted form of (2.5) is:-

$$c_i \theta(t) = \int_{\Omega(t)} w_i m(u) d\Omega, \quad i = 1, 2 \ldots N. \tag{2.7}$$

Assuming the c_i remain constant and applying Leibnitz's rule [60]:-

$$\begin{aligned} c_i \dot{\theta}(t) &= \frac{d}{dt} \int_{\Omega(t)} w_i m(u) d\Omega \\ &= \int_{\Omega(t)} \frac{\partial (w_i m(u))}{\partial t} d\Omega + \int_{\partial\Omega(t)} w_i m(u) \dot{\mathbf{x}} \cdot \hat{\mathbf{n}} dS \\ &= \int_{\Omega(t)} w_i m'(u) u_t d\Omega + \int_{\Omega(t)} m(u) \frac{\partial w_i}{\partial t} d\Omega + \int_{\partial\Omega(t)} w_i m(u) \dot{\mathbf{x}} \cdot \hat{\mathbf{n}} dS, \; i = 1, 2 \ldots N, \end{aligned} \tag{2.8}$$

where $\hat{\mathbf{n}}$ is the outward pointing, unit-length normal at any point on the surface of Ω.

Our w_i have been chosen to advect with velocity $\dot{\mathbf{x}}$ [4], therefore $\frac{\partial w_i}{\partial t} = -\dot{\mathbf{x}} \cdot \nabla w_i$, and hence:-

$$c_i \dot{\theta}(t) = \int_{\Omega(t)} w_i m'(u) u_t d\Omega + \int_{\Omega(t)} m(u)(-\dot{\mathbf{x}} \cdot \nabla w_i) d\Omega + \int_{\partial\Omega(t)} w_i m(u) \dot{\mathbf{x}} \cdot \hat{\mathbf{n}} dS, \; i = 1, 2 \ldots N. \tag{2.9}$$

Using the vorticity arguments in [4], equation(s) (2.9) can be used to solve uniquely for $\dot{\mathbf{x}}$, if $\dot{\theta}$ and u are known and $\nabla \times \dot{\mathbf{x}}$ is specified. In fact, we specify $\nabla \times \dot{\mathbf{x}}$ as zero, so that $\dot{\mathbf{x}}$ can be written as $\dot{\mathbf{x}} = \nabla\phi$, for a velocity potential ϕ. We also assume $\dot{\mathbf{x}} \cdot \hat{\mathbf{n}}$ is known at the boundary ($= \dot{\xi} \cdot \hat{\mathbf{n}}$ say). We will address this issue, along with that of finding $\dot{\theta}$ shortly, but first, writing $\dot{\mathbf{x}} = \nabla\phi$ in equation (2.9) and using the original PDE (2.1), we have:-

$$c_i \dot{\theta}(t) + \int_{\Omega(t)} m(u) \nabla\phi \cdot \nabla w_i d\Omega = \int_{\Omega(t)} w_i m'(u) L u d\Omega + \int_{\partial\Omega(t)} w_i m(u) \dot{\xi} \cdot \hat{\mathbf{n}} dS, \; i = 1, 2 \ldots N. \tag{2.10}$$

In discrete terms, if we approximate ϕ by $\Sigma_{i=1}^{N} \phi_i w_i$, then we now have N equations in $N+1$ unknowns, because $\dot{\theta}$ is unknown as well as the ϕ_i. However we can add an equation for $\dot{\theta}$ by developing equation (2.5) in a similar manner to above, to arrive at:-

$$\dot{\theta}(t) = \int_{\Omega(t)} m'(u) L u d\Omega + \int_{\partial\Omega(t)} m(u) \dot{\xi} \cdot \hat{\mathbf{n}} dS. \tag{2.11}$$

However, these $N+1$ equations are actually singular as they stand, as equation (2.11) is effectively a sum of all the equations in (2.10). This can readily be rectified by eliminating one of the ϕ_i, say ϕ_κ and the corresponding equation (with w_κ). Without loss of generality, we will actually set $\phi_1 = 0$, as it is only $\nabla\phi$ that is of interest.

As already noted, the above argument assumes that we have a means of calculating or estimating $\dot{\mathbf{x}} \cdot \hat{\mathbf{n}}$ on the boundary. This is discussed in the next Section, but before this we

complete our discussion of the algorithm.

Having found ϕ, we then calculate $\dot{\mathbf{x}}$ from the weighted form of $\dot{\mathbf{x}} = \nabla\phi$:-

$$\int_{\Omega(t)} w_i \dot{\mathbf{x}} d\Omega = \int_{\Omega(t)} w_i \nabla\phi d\Omega, \quad i = 1, 2 \ldots N. \tag{2.12}$$

We now use these values of $\dot{\mathbf{x}}$ to calculate $\dot{u}$ from (2.4), and hence return both $\dot{\mathbf{x}}$ and $\dot{u}$, so completing the algorithm.

The complete algorithm is:

1. From initial values of u and $\mathbf{x}$, calculate the monitor total θ and the monitor distribution constants c_i from (2.5) and (2.6).

2. Solve equations (2.10) and (2.11) to find $\dot{\theta}$ and ϕ (though $\dot{\theta}$ is not normally used, unless we also wish to keep track of $\theta(t)$ as the solution progresses).

3. Solve equation (2.12) to find $\dot{\mathbf{x}}$.

4. Solve equation (2.4), using $\dot{\mathbf{x}}$ (and current values of u and $\mathbf{x}$), to find $\dot{u}$.

5. Update (with Forward Euler timestepping[1]) u and $\mathbf{x}$, from $\dot{u}$ and $\dot{\mathbf{x}}$, and so return to step 1.

This can be described more simply as:-

BHJx algorithm

Loop:

$u \rightarrow (\theta, c_i)$

$(u, c_i) \rightarrow \phi \rightarrow \dot{\mathbf{x}}$.

$(u, \dot{\mathbf{x}}) \rightarrow \dot{u}$.

[1] In this thesis we only use Forward Euler for simplicity. The extension to any explicit Runge-Kutta or multistep scheme is straightforward [4] however. We have not considered implicit time-stepping schemes.

Update $\mathbf{x}$ and u.

End Loop.

We can compare this with the BHJ algorithm:-

Once only: Calculate θ, c_i

Loop:

$\theta * c_i \rightarrow u$.

$u \rightarrow (\phi, \dot{\theta}) \rightarrow \dot{\mathbf{x}}$.

Update θ and $\mathbf{x}$.

End Loop.

We can see the stage to calculate $\dot{\mathbf{x}}$ is essentially the same, but in BHJ, we recover u directly from an updated $\mathbf{x}$ and θ, whereas in BHJx, a form of the ALE is used to calculate $\dot{u}$ from $\dot{\mathbf{x}}$, and then u and $\mathbf{x}$ are updated.

The foregoing describes the BHJx algorithm in its most general form. However, there were two further amendments made for use in this thesis, following early development of the algorithm:-

- When solving equations (2.10) and (2.11), there was an accuracy improvement if ϕ was assumed to be zero on all boundary nodes, for the test cases considered. This is equivalent to assuming the tangential component of $\dot{\mathbf{x}}$ is zero at the boundary [4].

- All the problems solved in this thesis have Dirichlet boundary conditions, with the dependent variable u being zero on the boundary. To improve robustness, this has been enforced strongly by only solving equation (2.4) for internal nodes, with $\dot{u}$ set to zero for the boundary nodes. See Section 2.2 for more details on this.

Unless otherwise stated, these amendments apply to all code runs in this thesis.

2.1.3 Normal Boundary Velocity

As noted in the previous section, the extended version of the BHJx algorithm requires an estimate of $\dot{\mathbf{x}} \cdot \hat{\mathbf{n}}$ on the boundary. This may be obtained in at least two different ways, both of which have been considered in this work.

In the following chapter (where monitor functions of the form $m(u) = u^{\gamma}$ are explored), we consider a particular case where $\dot{\mathbf{x}}$ is known analytically on the boundary, in this case zero. However, the principle is identical whenever $\dot{\mathbf{x}}$ is known explicitly or is a computable function of the solution u and/or its derivatives.

In the subsequent chapters, we solve problems for which we assume $\dot{\mathbf{x}} \cdot \hat{\mathbf{n}}$ is not known on the boundary. In these cases, it is necessary to approximate it, based upon the use of a mass monitor function. In these later chapters, monitors of the form $u + a$ and $\sqrt{(}1 + (u_x)^2)$ are considered. Note that in this latter case, $m(u)$ is replaced by $m(u_x)$, so a further generalization of the method is described at that point.

A third possibility is to treat $\dot{\mathbf{x}} \cdot \hat{\mathbf{n}}$ on the boundary as another unknown in the algorithm, so it is solved for along with the internal mesh velocities. This was the method used in our development stage detailed in Section 2.2 for the mass monitor $m(u) = u$. However, such an approach has proved unreliable for general monitors ($m(u) \neq u$), and we do not discuss it further here.

2.2 BHJx - Development History

This section provides a description of the development of the BHJx algorithm, focusing entirely on the case $m(u) = u$, as used in [4]. In addition to providing justification for decisions made, the results presented also act as validation of the software that has been developed. We repeat here, for convenience, the BHJ algorithm described in Section 1.8, in its simplest form, together with two of the principal equations:-

Once only: Calculate θ, c_i

Loop:

$\theta * c_i \rightarrow u$.

$u \rightarrow (\phi, \dot{\theta}) \rightarrow \dot{\mathbf{x}}$.

Update θ and $\mathbf{x}$.

End Loop.

$$\theta(t) = \int_{\Omega(t)} u d\Omega. \tag{2.13}$$

$$c_i \theta(t) = \int_{\Omega(t)} w_i u d\Omega, \quad i = 1, 2 \ldots N. \tag{2.14}$$

If we simply replace u with $m(u)$ in equations (2.13) and (2.14), then it is still possible to develop (2.14) to get mesh velocities relating to the general monitor function $m(u)$ as shown in the previous section. In fact, the mathematical logic in obtaining $\dot{\mathbf{x}}$ in BHJx is almost identical to that in BHJ. There is a problem though, in trying to recover u from (2.14) if u is replaced there by $m(u)$, and $m(u)$ is non-linear. So what was experimented with is other ways of using $\dot{\mathbf{x}}$ to update u. Two methods were tried, both using an Arbitrary Lagrangian-Eulerian (ALE) form of the PDE [5]. The first method uses a weak differential form of the ALE. This was eventually the one selected, being equation (2.4) in Section 2.1:-

$$\int_{\Omega(t)} w_i \dot{u} d\Omega = \int_{\Omega(t)} w_i (Lu + \nabla u \cdot \dot{\mathbf{x}}) d\Omega, \quad i = 1, 2 \ldots N, \tag{2.15}$$

where w_i are N piecewise linear basis functions, which form a partition of unity [4]. This will be referred to in this section as the *non-conservative* ALE as it may not conserve mass (u). As described in Section 2.1, equation (2.15) is used to effectively convert $\dot{\mathbf{x}}$ to $\dot{u}$, so then $\mathbf{x}$ and u can be updated in a timestepping system.

The second method uses a weak *integral* form of the ALE [5]:-

$$\frac{d}{dt}\int_{\Omega(t)} w_i u d\Omega = \int_{\Omega(t)} w_i (Lu + \nabla\cdot(u\dot{\mathbf{x}}))d\Omega, \quad i = 1,2\ldots N. \tag{2.16}$$

This will be referred to in this section as the *conservative* ALE as it will conserve mass (u) under certain conditions, as explained below. If we write (using ϑ to distinguish this from the *monitor* total θ):-

$$\vartheta_i = \int_{\Omega(t)} w_i u d\Omega, \tag{2.17}$$

then equation (2.16) allows us to calculate $\dot{\vartheta}_i$ from $\dot{\mathbf{x}}$ and update ϑ_i at each timestep. The new values of u can then be recovered from (2.17). So a second algorithm was tried in the development stage, using the conservative rather than the non-conservative ALE to obtain an updated u from $\dot{\mathbf{x}}$. We will refer to this as BHJx(c). In practice, this algorithm is essentially the same as BHJx, but instead of using (2.15) to update u, we use (2.16) and (2.17). We can describe it more simply as:-

BHJx(c) algorithm

Once only: Calculate initial values of ϑ_i

Loop:

$\vartheta_i \rightarrow u$.

$u \rightarrow (\theta, c_i)$

$(u, c_i) \rightarrow \phi \rightarrow \dot{\mathbf{x}}$.

$\dot{\mathbf{x}} \rightarrow \dot{\vartheta}_i$.

Update $\mathbf{x}$ and ϑ_i.

End Loop.

Both the BHJx and BHJx(c) algorithms started with weak (zero) boundary conditions enforced, so that appropriate boundary integrals are disregarded, as described in Section 1.8. Hence the calculation of $\int_{\Omega(t)} w_i L u d\Omega$ is simplified when calculating $\dot{u}_i$ for BHJx in equation (2.15) and $\dot{\vartheta}_i$ for BHJx(c) in equation (2.16). Note that neither algorithm strictly

enforces the "distribution constants" c_i to be constant, as was the case in the original BHJ algorithm. This does not prevent the algorithm from functioning, but it does leave an open question as to whether the distribution is being maintained. In the case of the initial conditions being optimised (equidistributed for the monitor), this question equates to whether the monitor is actually being equidistributed for all time. This question will be addressed later in this thesis.

A comparison of the conservative and non-conservative form of the ALE, made by Baines, Hubbard, Jimack and Jones when studying the porous medium equation, shows that the conservative form is "preferable both in terms of accuracy and robustness" [5]. At the time of development of the BHJx algorithm, research showed that when using the conservative ALE, strong boundary conditions (where u is forced to be zero at the boundary) gave more accurate results than weak ones, again studying the porous medium equation [51]. However, that same research also shows that this combination of strong boundary conditions and conservative ALE enforces mass conservation. In particular, for a PDE where the spatial operator is of the form $\nabla \cdot \mathbf{F}$, mass will be conserved if [51]:-

$$(\mathbf{F}u + u\dot{\mathbf{x}}) \cdot \hat{\mathbf{n}} = 0 \tag{2.18}$$

on the boundary, where $\hat{\mathbf{n}}$ is the outward pointing unit-length normal to $\partial\Omega$. Whilst this may be a key property for some numerical PDE algorithms, it could also be considered unnecessarily restrictive for general PDE study. Therefore, both forms of the ALE were studied, so that both BHJx and BHJx(c) were considered, and these with both weak and strong boundary conditions. This and the need to start from the solid foundation that formed the original BHJ paper led to the code development matrix shown in Table 2.1 (the acronym "NLP" stands for Non-Linear Parabolic).

All these codes were written in "C++" and model 2D problems. The PME problem, which is fully described in Section 1.6.1, is:-

Code	PDE	Algorithm	Boundary Conditions	Comments
BHJ1	PME	BHJ	Weak	
BHJ2	PME	BHJ	Weak	SPARSKIT added
BHJ3	Oxygen	BHJ	Weak	
NLP1	Oxygen	BHJx	Weak	
NLP2	PME	BHJx(c)	Weak	
NLP3	PME	BHJx	Strong	Weak tried initially
NLP4	PME	BHJx(c)	Strong	Uses Petrov-Galerkin method

Table 2.1: Code development matrix

$$\frac{\partial u}{\partial t} = \nabla \cdot (u^n \nabla u)\ (\mathbf{x} \in \Omega, t > 0), n \text{ being a positive integer; } u \Big|_{t=0} = u_0(\mathbf{x}); u \Big|_{\partial\Omega} = 0. \quad (2.19)$$

The oxygen PDE models diffusion of oxygen in an absorbing medium, such as tissue [4, 8]. This can be defined on a moving boundary $\Omega(t)$ as follows:-

$$\frac{\partial u}{\partial t} = \triangle u - 1\ (\mathbf{x} \in \Omega, t > 0);\ u \Big|_{t=0} = u_0(\mathbf{x}); u \Big|_{\partial\Omega} = 0 = \nabla u \cdot \hat{\mathbf{n}} \Big|_{\partial\Omega}. \quad (2.20)$$

We now detail the development history of these codes. The first code, BHJ1, was a straighforward conversion from the Fortran code used to study the porous medium equation in the original BHJ paper [4]. The first test was to get agreement with that code, running to 0.2s on a 545-node mesh, with a timestep of 0.0001 and an initial radius of 0.5. The initial conditions for these runs were taken to be the known solution in equations (1.26) and (1.27) (this will usually be the case in this thesis when testing for convergence). The initial meshes are shown in Figure 2.1.

The BHJ1 final mesh differed from the Fortran one by no more than 1×10^{-10} in values of the mesh co-ordinates x, y and the dependent variable u, so the basic Fortran to 'C++' conversion was considered complete. This first program used a simplified version of the BHJ algorithm, utilising the fact that the PME was mass-conservative. So since $\dot{\theta} =$

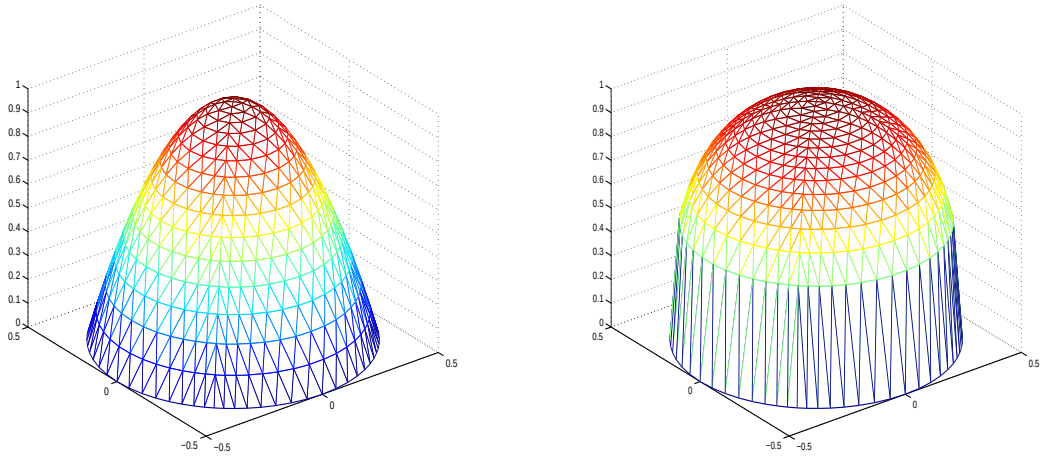

Figure 2.1: Porous medium equation: initial conditions for n=1 (left) and n=3 (right).

0, only a symmetric system needed to be solved when finding ϕ from equations (1.39) and (1.40). To allow for later development then, the symmetric-only system solver was replaced with a more general-purpose one, using the SPARSKIT library [83], so forming the BHJ2 program. Comparing this to BHJ1 with the same mesh and conditions as above, there was a difference in the final mesh of BHJ2, as compared to BHJ1, of no more than 2×10^{-9} in the values of the mesh co-ordinates x, y and the dependent variable u.

For the last part of these foundation codes, the PME problem was replaced by the oxygen problem in BHJ2, to become BHJ3. This is a non-conservative problem, so is a further test of the SPARSKIT addition. Furthermore, this problem is also used to provide a first test of the BHJx algorithm, which is based upon the non-conservative ALE method (implemented in the code NLP1). As this has no known analytic solution in 2D, the testing here has to be partly qualitative. The initial conditions are shown in Figure 2.2. Following [4], this mesh (of 615 nodes) was created using the known solution for 1D ($e^{x-1} - x$ for $x \in [0,1]$), which at $t = 0$ is $e^{r-1} - r$, where r is the radius, with the initial boundary set at $r = 1$. The graphs in the top-right and bottom-left (respectively) in Figure 2.2 show final meshes for BHJ3 and a Fortran code supplied as part of the development

of the BHJ paper [4], both of these running to $T = 0.07$ with a timestep of $0.0001s$. We can see a general agreement between the codes.

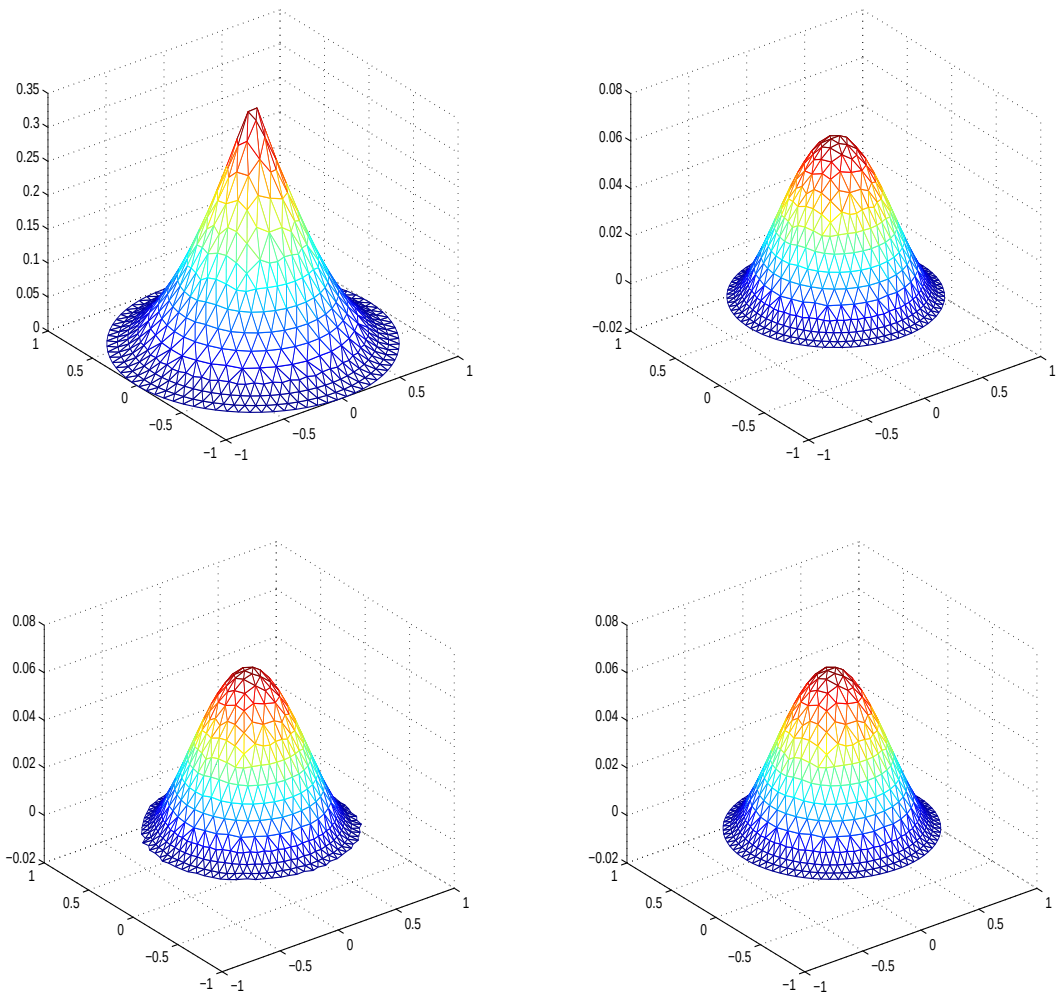

Figure 2.2: Oxygen problem: initial conditions (top left), BHJ3 final mesh (top right), Fortran code final mesh (bottom left), NLP1 final mesh (bottom right).

We can also show an extrapolated convergence for BHJ3. In Figure 2.3, we show the graph of $\theta(t)$ (monitor total) for the same 615-node grid as above, but with decreasing timesteps. We can see a convergence to some solution in the graph - this is clearer in the zoomed graph, as the plots have effectively merged in the main graph. $\theta 1$ refers to $\theta(t)$ for $dt = 0.0001$, and so on.

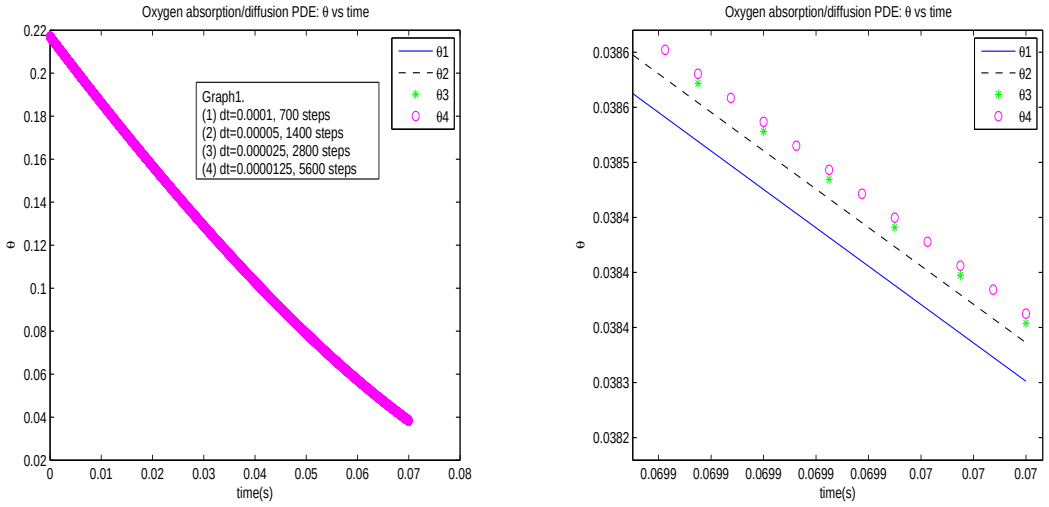

Figure 2.3: Oxygen problem: $\theta(t)$ and zoom in (BHJ3).

The NLP1 code was developed next - this also studies the oxygen problem, but with the BHJ algorithm in BHJ3 replaced with the non-conservative ALE method used in BHJx. This was a first test of BHJx in its most general form - the non-conservative ALE with a problem known to lose mass. The first run was done with exactly the same conditions as for the BHJ3 run above, and the final mesh is shown in Figure 2.2 (bottom-right) for a qualitative comparison with BHJ3. In addition to looking equivalent, a direct numerical comparison of the final BHJ3 and NLP1 meshes was done - this showed a difference of no more than 0.4%, when looking at the mesh co-ordinates x,y and the dependent variable u, this being the absolute difference of the two values, divided by the maximum of them, i.e., $\text{diff}(x1,x2) = \frac{|x1-x2|}{\max(|x1|,|x2|)}$. This result adds confidence to NLP1, from the point of view of a solid foundation. Extrapolated convergence tests were also done here, using the same conditions as BHJ3, but to a longer run time of 0.09s - the plots of $\theta(t)$, as dt decreases, are shown in Figure 2.4, where we again see convergence to a solution.

These results for NLP1 gave sufficient confidence to move to the final testing phase - trying out the alternative conservative ALE, and quantifying the effects of weak vs strong boundary conditions. This was done with the codes NLP2, 3 and 4, all modelling the

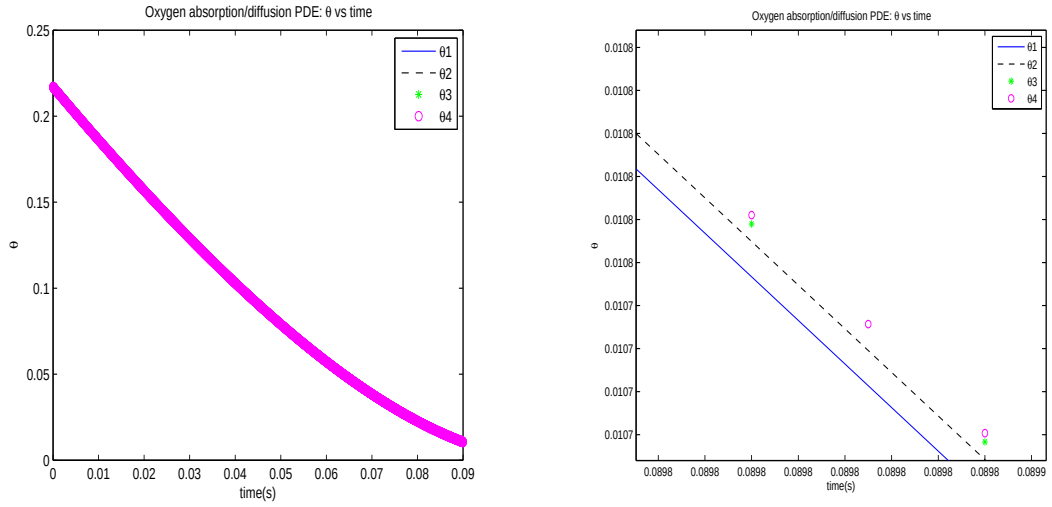

Figure 2.4: Oxygen problem: $\theta(t)$ and zoom in (NLP1).

porous medium equation (the rest of the details are shown in Table 2.1). For testing of BHJx with the PME, we first note that NLP3 runs failed early with weak boundary conditions, for example, one 2113-node mesh with $n = 1$ tangled (had negative element areas) shortly after $T = 0.05$, so that code was amended to have strong boundary conditions, and all comparisons from here on regarding NLP3 are with the strong-BCs version. A comparison is shown in Figure 2.5 of NLP3 with BHJ2 (which has weak boundary conditions), for a 545-node mesh, initial radius 0.5, timestep of 0.0001, running to T=2.0, for both n=1 and n=3.

We can see both codes have completed their runs, and show a broadly similar result, except for the boundary. The "lifting" effect seen in the BHJ2 graphs is actually caused by the base plane of the graph being drawn at a lower "z-value" for these meshes, since the values of u at the boundary were actually a mix of small positive and negative values, due to boundary conditions being only weakly enforced.

Finally, we look at a study of the alternative (conservative ALE) BHJx(c) algorithm, which also incorporates a weak/strong boundary conditions comparison. The NLP2 code was created from NLP1, by changing the PDE to the porous medium equation, and the

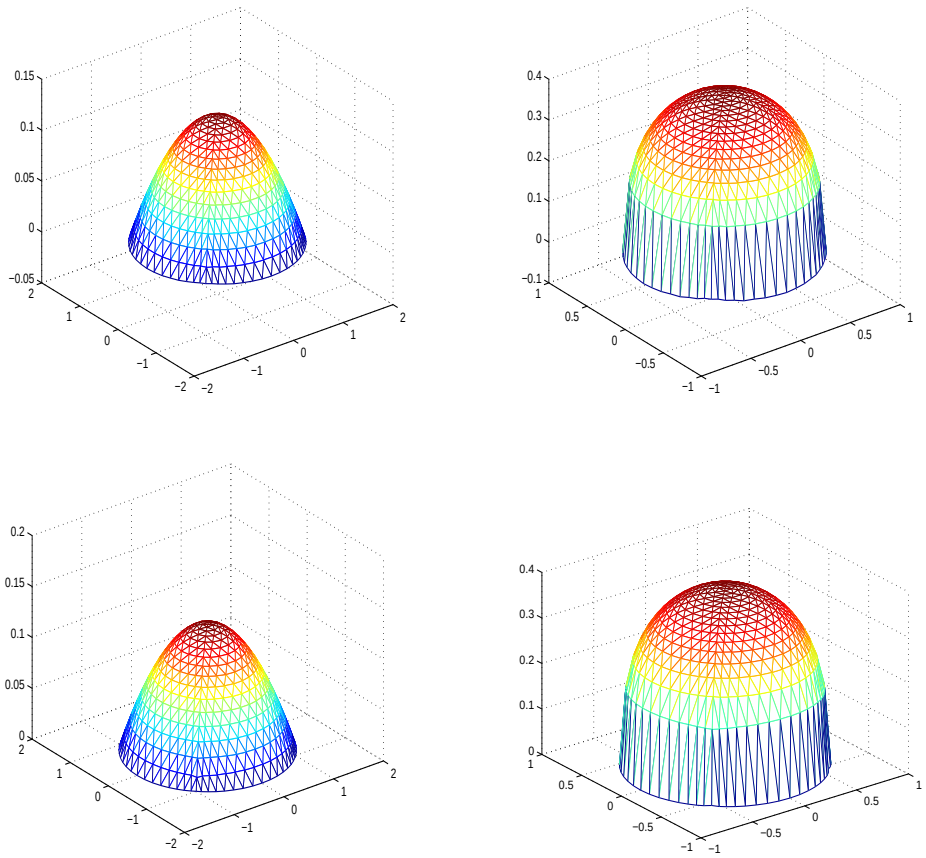

Figure 2.5: Porous medium equation: BHJ2 (n=1) final mesh (top left), BHJ2 (n=3) final mesh (top right), NLP3 (n=1) final mesh (bottom left), NLP3 (n=3) final mesh (bottom right).

algorithm from BHJx to BHJx(c), keeping the boundary conditions weak. NLP4 was created from NLP2 by making the boundary conditions strong. Following [51], this was done in a "consistent" manner, so that mass conservation was definitely enforced. This meant having a test space different from the trial space (so that the test functions form a partition of unity), hence this code is using a Petrov-Galerkin approach (see [51] for more details). As the porous medium equation has a known solution, we can estimate accuracy of these codes. Orders of convergence for NLP2, 3 and 4 are shown in Figure 2.6, for $n = 1$ and $n = 3$, meshes (initial radius 0.5) of 545, 2113 and 8321 nodes (so a dx decreasing by 50%), and a timestep of 0.0001 for 545 nodes (and reducing to 25% for

each mesh).

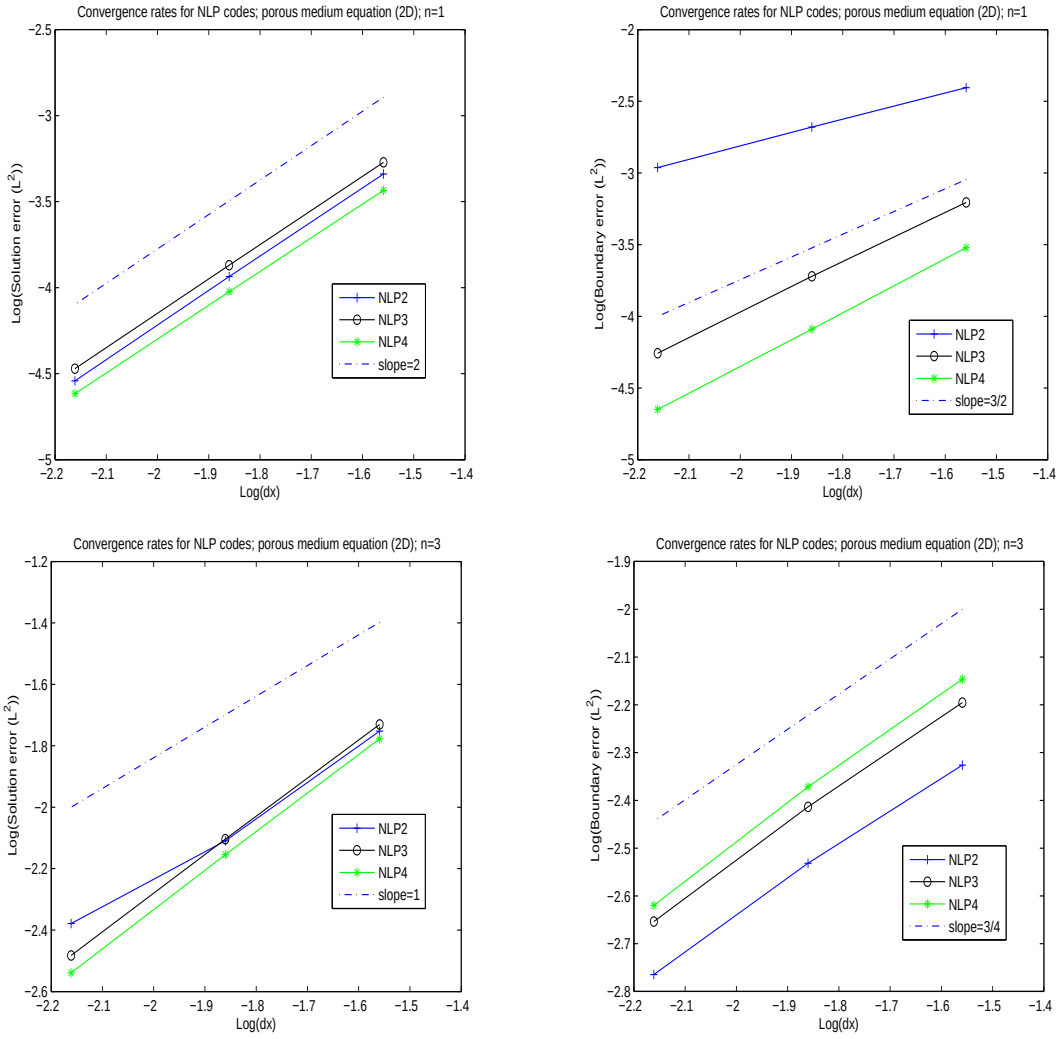

Figure 2.6: Porous medium equation. Orders of convergence for: L^2 mesh error, n=1 (top left), L^2 boundary error, n=1 (top right), L^2 mesh error, n=3 (bottom left), L^2 boundary error, n=3 (bottom right).

We can see that NLP4 is generally the most accurate and has an improved order of accuracy over NLP2, when considering the boundary error, and this broadly agrees with the findings in [51]. We also note the orders of convergence, over all four cases (so all four graphs in Figure 2.6) for NLP4, are comparable to the original BHJ study [4]. We can see also though, that the order of convergence of NLP3 is as good as NLP4, so as the NLP4 algorithm is more restrictive, as it may unnecessarily enforce mass conservation, NLP3

was taken forward as the candidate for more general monitors. Note that all the results in this section, considering both the Oxygen problem and the PME, show that BHJx with the mass monitor gives broadly similar results to BHJ, therefore any comparisons of BHJx (non-mass monitor) with BHJx (mass monitor) are broadly equivalent to comparing BHJx (non-mass monitor) with BHJ.

Chapter 3

The blow-up problem

We consider here the semilinear heat equation, with a fixed boundary and Dirichlet boundary conditions:-

$$u_t = \triangle u + u^p, \mathbf{x} \in \Omega, t > 0; u\Big|_{t=0} = u_0(\mathbf{x}); u\Big|_{\partial\Omega} = 0,\ p \text{ is a positive integer.} \qquad (3.1)$$

This equation describes the temperature of a reacting medium, such as a burning gas [20].

3.1 Background

No analytical solution of (3.1) is known to this author, though there are conditions given by Weissler [94] for solution existence when the domain is all of $\mathbb{R}^n$ and $p > 1$. In that case, he shows that under mild restrictions, if $n(p-1)/2 \leq 1$, non-negative L^p solutions always blow-up in L^p norm in finite time, and if $n(p-1)/2 > 1$, global solutions exist, given sufficient conditions on the initial data. For example, we will have blow-up in a finite time in the 1D case, if $p = 2$ or 3, and in 2D, if $p = 2$.

In this chapter, we consider a 1D version of (3.1) with a finite interval:-

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

$$u\Big|_{t=0} = u_0(x); u\Big|_{x=0} = 0; u\Big|_{x=1} = 0, \ p \text{ is a positive integer.} \tag{3.2}$$

In this case, any solution of (3.2) will blow up at a single point x^*, at a time T, if u_0 is "sufficiently large" [20], so that as $t \to T$, u develops a narrowing peak around x^*. Although we do not have an analytical solution for (3.2), there is some asymptotic knowledge. In particular, for x near to x^*, if we define the "kernel co-ordinate"μ [20, 48] by:-

$$\mu(x,t) = (x - x*)[(T-t)(\alpha - log(T-t))]^{-\frac{1}{2}}, \tag{3.3}$$

where α is a constant depending on the initial conditions, then where μ is constant, the solution $u(x,t)$ of (3.2) follows this asymptote [20]:-

$$u(x,t) \to \frac{\beta^\beta [1 + \mu^2/4p\beta]^{-}\beta}{(T-t)^\beta} \text{ as } t \to T, \tag{3.4}$$

where $\beta = \frac{1}{p-1}$.

This asymptotic behaviour has been confirmed in a numerical study by Budd, Huang and Russell [20], using MMPDE methods. The equation exactly as in (3.2) was studied in 1D, with $p = 2$, the monitor function $m(u) = u$ and $u_0 = 20sin(\pi x)$. Values of x^* and T have actually been calculated analytically by using the asymptotic theory and the MMPDEs, but we should emphasise this was possible because of using this particular method of solution, and so depend on that method. This study used an adaptive timestep, and we note that papers by Budd and Williams [19], and Ceniceros and Hou [26] state that adaptive timestepping is necessary to solve the semilinear heat equation, with the latter giving a specific formula for the adaptation:-

$$dt = dt_0/||pu^{(p-1)}||_\infty, \tag{3.5}$$

where dt_0 is the initial timestep.

The results of this study do show the blow-up happening at a single point x^* at a time T, with x^* being the peak point of the initial data[1]. This result has also been confirmed by studies of this problem at the University of Reading [28, 91].

3.2 Applying the BHJx algorithm

The algorithm is applied as in Section 2.1. As we have a fixed boundary, the boundary velocity ($\dot{\xi}$) is set to zero in equations (2.10) and (2.11). The monitor functions have been of the form $m(u) = u^\gamma$, as scale-invariance and similar studies [20, 48] have shown this form allows the mesh to evolve to correctly follow the asymptotic form, without ceasing to evolve at any point.

3.3 Preliminary results

The first cases have been run with the same initial and boundary conditions and mesh as in the Budd, Huang and Russell study [20], so that we have the initial conditions $u_0 = 20sin(\pi x)$, with the interval $[0.0, 1.0]$ split into 40 equally-spaced intervals between 41 nodes, and u being forced to be zero at the boundary nodes. Following that study, the monitor function $m(u) = u$ has been used for the $p = 2$ case, and $m(u) = u^2$ for $p = 3$. The initial conditions and a plot of $u_{max}(t)$ for the $p = 2$ run are shown in Figure 3.1, whilst plots of $u(x)$ for $p = 2$ and $p = 3$ at the algorithm breaking point of T are shown in Figure 3.2.

[1] In a 2D study by Budd and Williams [17] of the semilinear heat equation, on a circle and square, we also see a single blow-up point x^* at a time T, though T is not given. As in the study by Budd, Huang and Russell [20], we see the blow-up point x^* is at the peak of the initial data.

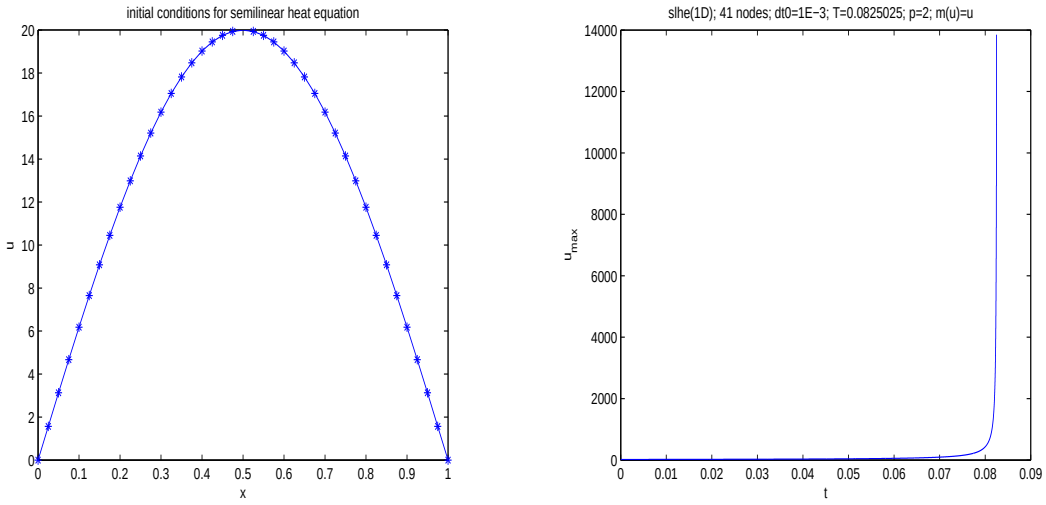

Figure 3.1: Semilinear heat equation: initial grid (left), and $u_{max}(t)$ for $p = 2, m(u) = u$, 41 nodes.

As predicted in [20], we get a single, narrowing peak round x^* (0.5 in both these cases), with $u \to \infty$ as $t \to T$, and x^* is also the peak of the initial data.

The $p = 2$ run has reached a point T=0.0825025 with $u_{max} = 14,000$ before tangling starts. The circled node in Figure 3.2 shows the tangling point, and the zoom shows the nodes about to overlap. This run did continue until T=0.0825201, with $u_{max} = 18,000$, but the algorithm then broke down. For $p = 3$, the algorithm broke down immediately after T=0.00128295 with $u_{max} = 3142$. There was no tangling here, but the nodes are very closely centred round x^*, as the zoomed plot shows - this may be the cause of the problem. We will discuss untangling and restarting these runs in Section 3.6, but from these first runs, we note:-

- The "pre-tangle" time of T=0.0825025 for the $p = 2$ case is comparable with the blow-up times of T=0.082291 (MMPDE4 method) and T=0.082283 (MMPDE6 method) of [20].

- These cases were run with adaptive timestepping, using the formula in (3.5). This produced no significant difference in T or $u_{max}(T)$, but it did reduce the program

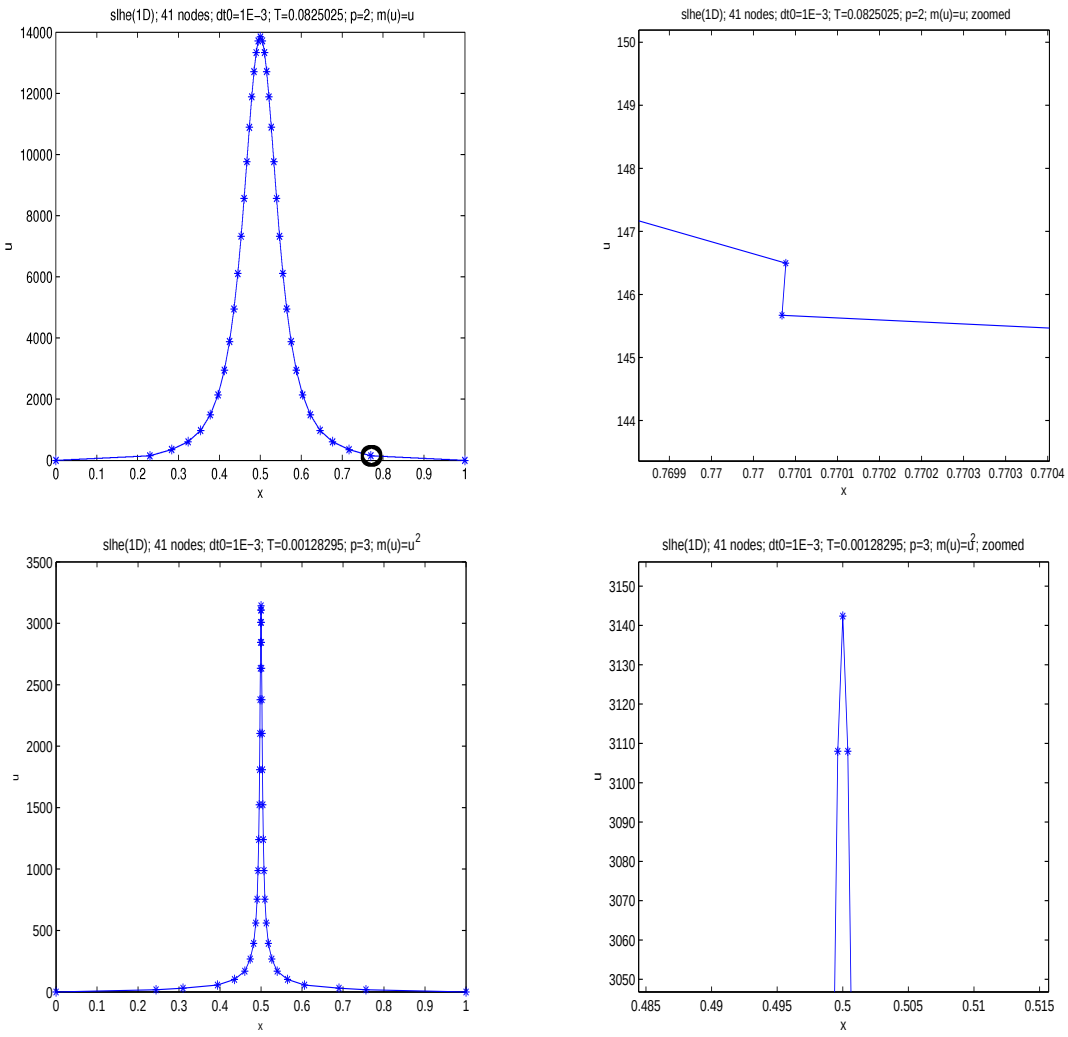

Figure 3.2: Semilinear heat equation (41 node grid): solutions at the time when the algorithm breaks down, for $p = 2$, $m(u) = u$ (top) and $p = 3, m(u) = u^2$ (bottom).

run-time, so that for the $p = 2$ case, an initial timestep of 1×10^{-5} with fixed timestepping gave broadly the same results as an initial timestep of 1×10^{-3} with adaptive timestepping. With adaptive timestepping, reducing the initial timestep of 1×10^{-5} had no benefit, and increasing it to more than 1×10^{-3} caused the algorithm to fail very early on. We can see the benefit and justification for adaptive timestepping from the plot of $u_{max}(t)$ in Figure 3.1 - there is only slow growth in u_{max} until near T. All runs in this chapter were done with adaptive timestepping and

an initial timestep of 1×10^{-3}, unless otherwise stated.

- The values of $u_{max}(T)$ were 14,000 for $p = 2$ and 3142 for $p = 3$. The value for $p = 2$ is not as high as seen in [20], which is likely to be because the algorithm breakdown has prevented a higher value being reached.

3.4 Accuracy

The two cases in Section 3.3, namely $p = 2, 3$ with monitor u^{p-1} have been run to approximately half the run-time there, with a decreasing spatial interval, to demonstrate that the method is converging to something, though we do not have an analytical solution to compare with. For example, for $p = 2$ and $m(u) = u$, the values of u_{max} (which will be u at x^*) at t=0.04, for 21/41/81/161 nodes are 33.1364, 33.2578, 33.2906, 33.3030, to six significant digits. Continuing that sequence, based upon extrapolation, with an average of the rates of decrease of successive steps, gives us a nominal limit of 33.3062. With that as the limit, the error of the solution is then 0.1698, 0.0484, 0.0156, 0.0032. This has been plotted against dx, as a log-log graph, in Figure 3.3. We have also plotted the same results for $p = 3$, running to $t = 0.0006$, and to ensure this basic result is not dependent on the monitor, we have also run the $p = 2$ case with monitor $m(u) = u^2$, and the $p = 3$ case with monitor $m(u) = u$ to ensure the same limit is reached, which did indeed turn out to be the case[2]. Log-log plots of error against dx have been added to the graph for these last two cases. We can see from Figure 3.3 that we have a convergence rate of approximately second order for both $p = 2$ and $p = 3$, independently of the monitor, and that for $p = 2$, the actual errors are almost coincident, for the two monitors used.

[2]For $p = 2$ and monitor function $m(u) = u^2$, the 161-node case did need a lower initial timestep of 1×10^{-4} for the algorithm to commence.

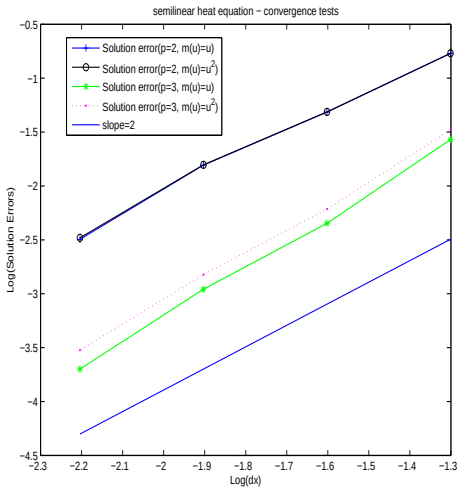

Figure 3.3: Semilinear heat equation - accuracy.

3.5 Other monitors

The two cases run at the start of Section 3.3 were run, exactly as stated there, but with different monitors, so that for $p = 2$, the monitor was $m(u) = u^2$, and for $p = 3$, the monitor was $m(u) = u$. These cases were again run until just before tangling starts, and the results are shown in Figure 3.4. We can see a lower peak of 202 was reached for the first case, but in the second case, a much higher peak of 2.87×10^7. The end times of these runs were T=0.0769416 for $p = 2$ and T=0.00128304071 for $p = 3$. After these times, both runs tangled. For the $p = 2$ case, this was again at the nodes next to the boundary. However, for $p = 3$, the tangling was at nodes near the spike. We can see then, that the algorithm progressed further, and so reached a higher u_{max} with the $m(u) = u$ monitor for both $p = 2$ and $p = 3$, rather than using $m(u) = u^{p-1}$.

3.6 Robustness

To investigate the tangling issue further, the $p = 2$ case with $m(u) = u$ monitor was run with 81 and 161 nodes (with all other conditions being the same as for the 41-node run in

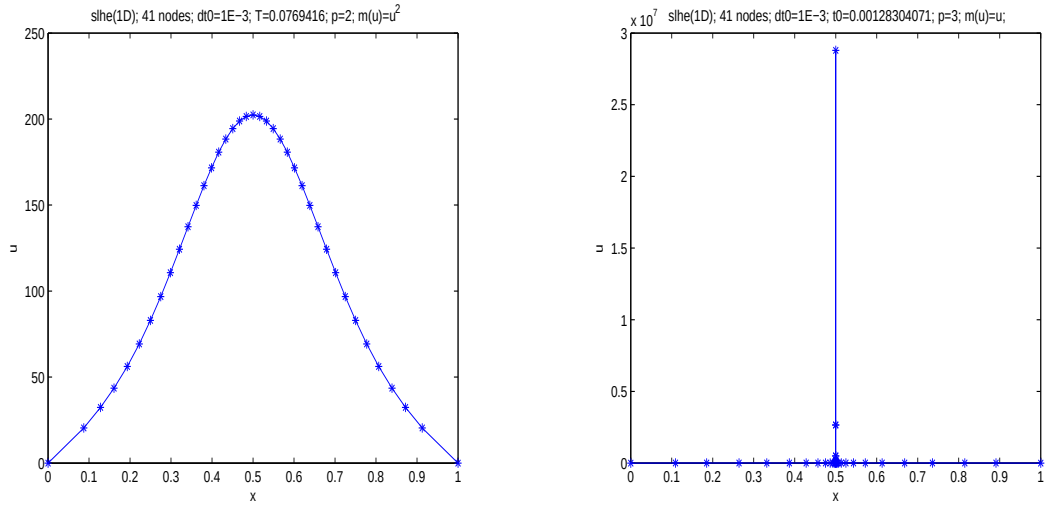

Figure 3.4: Semilinear heat equation (41 node grid): solutions at the time when the algorithm breaks down, for $p = 2$, $m(u) = u^2$ (left) and $p = 3, m(u) = u$ (right).

Section 3.3) until a failure in the algorithm. The 161-node run required a smaller initial timestep of 1×10^{-4} to run. The graphs of $u(x)$ at the breaking point T are shown in Figure 3.5. The 81-node run failed because of tangling at the next-to-boundary nodes, the 161-node run failed with a general algorithm breakdown, so that one of the matrix systems was unsolvable. The break points for 81 and 161 nodes were T=0.0824865 and T=0.0824442 respectively, which are comparable with T=0.0825025 for the 41-node case, but the values of u_{max} are higher, being 116,000 for 81 nodes, and 329,929 for 161 nodes, compared with 14,000 for 41 nodes, suggesting a higher value of u_{max} can be reached as $dx \to 0$, before some form of algorithm breakdown.

To further investigate the effect of tangling, we have taken the four main cases, i.e., 41 nodes with $p = 2, 3$ and monitors $m(u) = u$ and $m(u) = u^2$, and attempted to untangle and restart these runs, at the point of tangling or algorithm failure. The mesh at the pre-tangle point for $p = 2$ and $m(u) = u$, together with the untangled version, is shown in Figure 3.6. The untangling was achieved by equidistributing the tangled points to a (piecewise linear) shape defined by those points, using a monitor $u + a$, a being $u_{max}(T)$ (at the pre-tangle point). So in Figure 3.6, we see the untangled mesh has almost the same shape as the

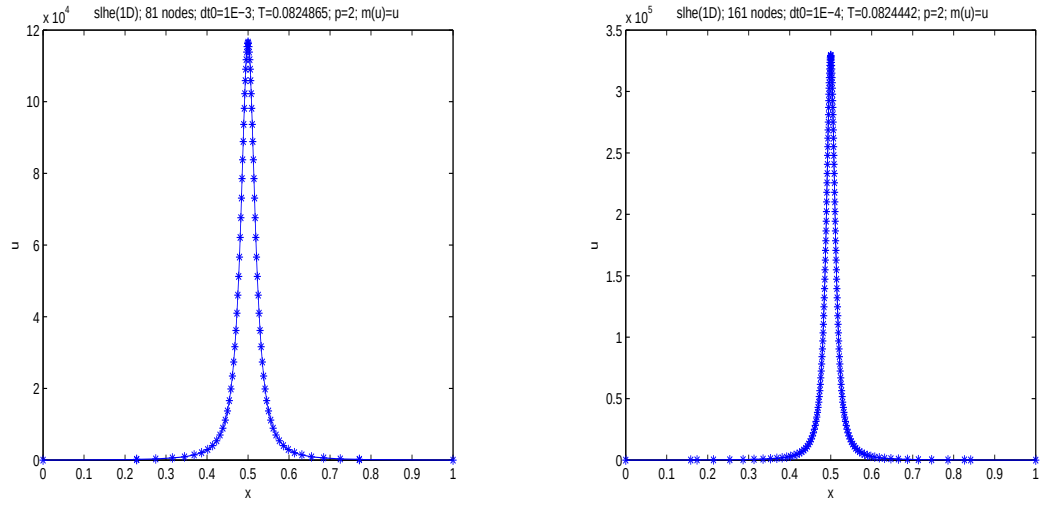

Figure 3.5: Semilinear heat equation: $u(x)$ for $p = 2$, $m(u) = u$, 81 and 161 nodes.

tangled one, but the points have been untangled, and also moved away from the spike.

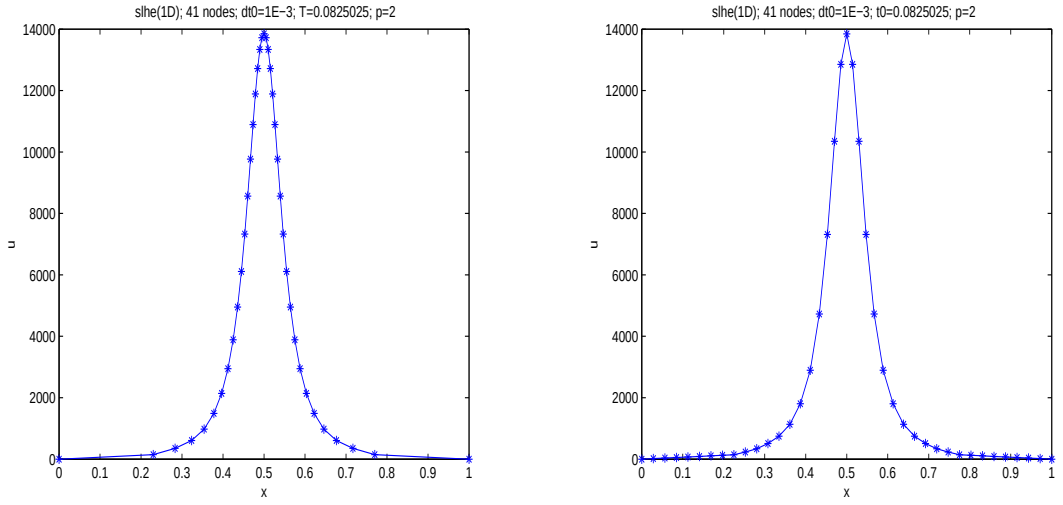

Figure 3.6: Semilinear heat equation: tangled mesh (left) and untangled mesh (right) for $p = 2$, $m(u) = u$.

The final meshes, following untangling and restarting, for $p = 2,3$ and $m(u) = u, u^2$ are shown in Figures 3.7 and 3.8, at a point just before further tangling or algorithm breakdown. For $p = 3$ and $m(u) = u$, the mesh would not restart - there was tangling at the first iteration, even with a reduced timestep. The two $p = 2$ cases needed a lower initial

timestep of 1×10^{-4} to start.

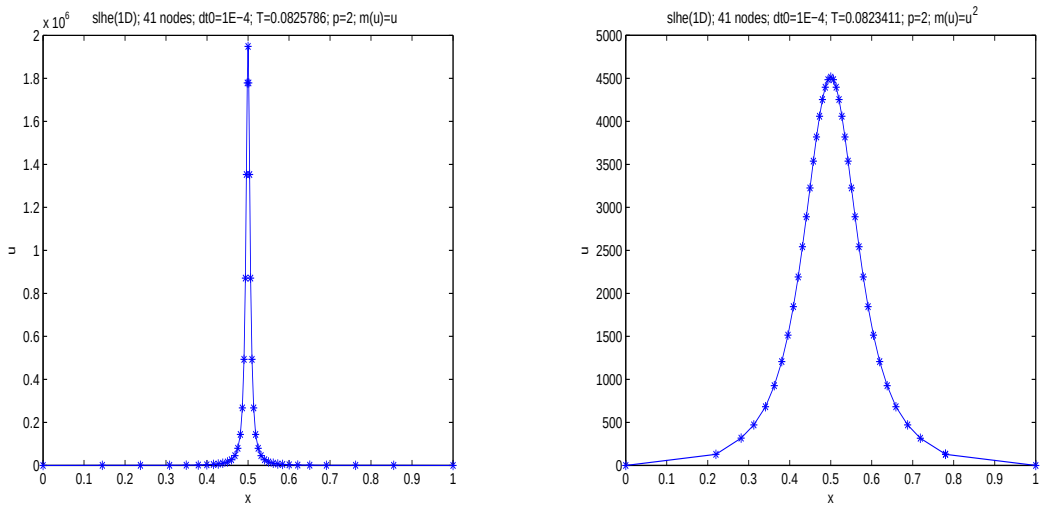

Figure 3.7: Semilinear heat equation: meshes following restarts for $p = 2$ and $m(u) = u$ (left) and u^2 (right).

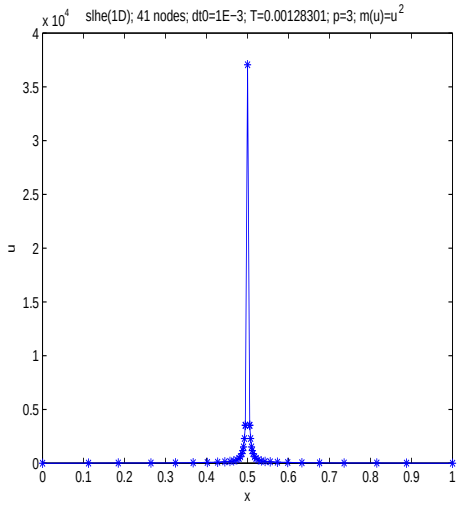

Figure 3.8: Semilinear heat equation: mesh following restart for $p = 3$ and $m(u) = u^2$.

The value of T was extended from 0.0825025 to 0.0825786, and the value of u_{max} extended from 14000 to 2×10^6 for $p = 2$ and $m(u) = u$. The corresponding increases for $p = 2$ and $m(u) = u^2$ and for $p = 3$ and $m(u) = u^2$ were $(0.0769416 \rightarrow 0.0823411, 202 \rightarrow 4511)$ and $(0.00128295 \rightarrow 0.00128301, 3142 \rightarrow 37059)$ respectively. No subsequent

restarts were attempted.

3.7 Comparison with a location-based method

The cases for $p = 2$ and $p = 3$ with the monitors $m(u) = u$ and $m(u) = u^2$ respectively, with the same initial conditions, and with 41 and 81 nodes, have been run with "movcol", a program based on an MMPDE method, described in Huang and Russell [44], allowing us to compare BHJx directly with a location-based method. The movcol program[3] was run until the algorithm broke down at a time T. A plot of $u_{max}(t)$ for the $p = 2$ case with 41 nodes is shown in Figure 3.9, with the initial conditions repeated here for clarity. We can see here the same pattern (as in BHJx) of very slow growth in u until we near the blow-up time T. For 41 nodes, both $p = 2$ and $p = 3$, movcol has reached higher values of $u_{max}(T)$, these being 980,486 and 29,500 for $p = 2$ and $p = 3$ respectively, compared with the BHJx values of 14,000 and 3,142 with the same monitor (u^{p-1}), though BHJx did achieve a $u_{max}(T)$ of 27,800,000 for $p = 3$ with the monitor $m(u) = u$ for the 41 node case. For 81 nodes, both $p = 2$ and $p = 3$, movcol again reached higher values of $u_{max}(T)$, these being 1,207,131 and 14,048 for $p = 2$ and $p = 3$ respectively, compared with the BHJx values of 116,000 and 2307, also with the same u^{p-1} monitor.

For 41 nodes, the movcol values of T for $p = 2$ and $p = 3$ are 0.0824369 and 0.00128093 respectively, which are comparable to the BHJx values of 0.0825025 and 0.00128295. For 81 nodes, the two movcol values for T are 0.0824363 and 0.00128093, which are comparable to the BHJx values of 0.0824865 and 0.00128295.

The final grids at the algorithm breaking point of T are shown in Figure 3.10 for both movcol 41-node cases, which can be compared with those for BHJx in Figure 3.2. We do not consider here any movcol runs with a monitor other than $m(u) = u^{p-1}$ for a given p, as the MMPDE study by Budd, Huang and Russell [20] shows that $m(u) = u^{p-1}$ is best

Case/Code	BHJx	movcol
p=2/41 nodes	15	132
p=2/81 nodes	225	7336
p=3/41 nodes	169	10
p=3/81 nodes	1552	176

Table 3.1: Efficiency comparison (CPU seconds) of BHJx and movcol, for semilinear heat equation

suited to the MMPDE method, when studying the semilinear heat equation.

CPU times (in seconds) for the four movcol cases run are shown in Table 3.1, along with those for BHJx. We can see BHJx is more efficient than movcol for the $p = 2$ case, but for $p = 3$, the reverse is true.

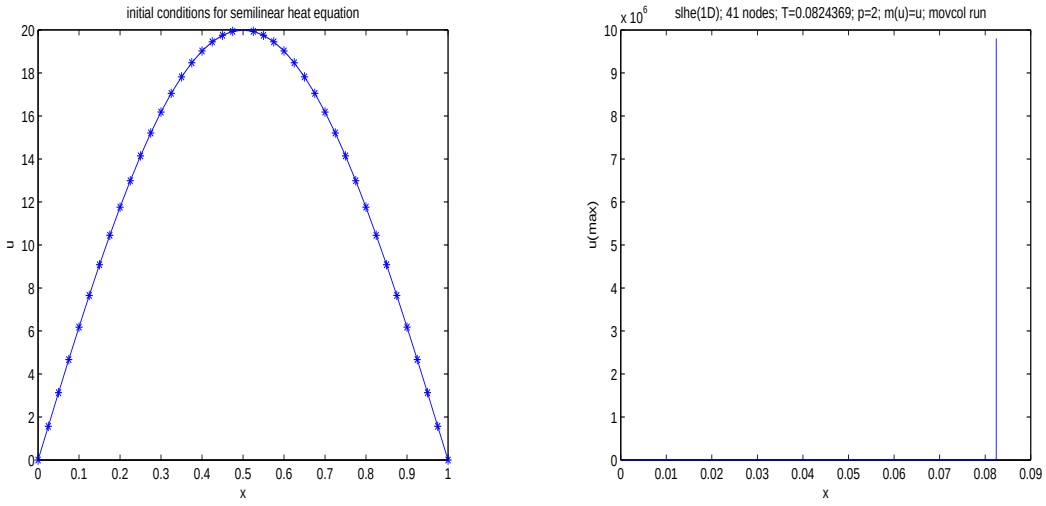

Figure 3.9: Semilinear heat equation: initial grid (left), and $u_{max}(t)$ for $p = 2, m(u) = u$, 41 nodes, movcol run.

3.8 Summary and Discussion

The preliminary results show the algorithm reflects the basic features of the analysis and the existing research, so that we see blow-up happening at a single point x^*, which is

[3]movcol is available for download at http://www.math.ku.edu/ huang/research/movcol/movcol.html.

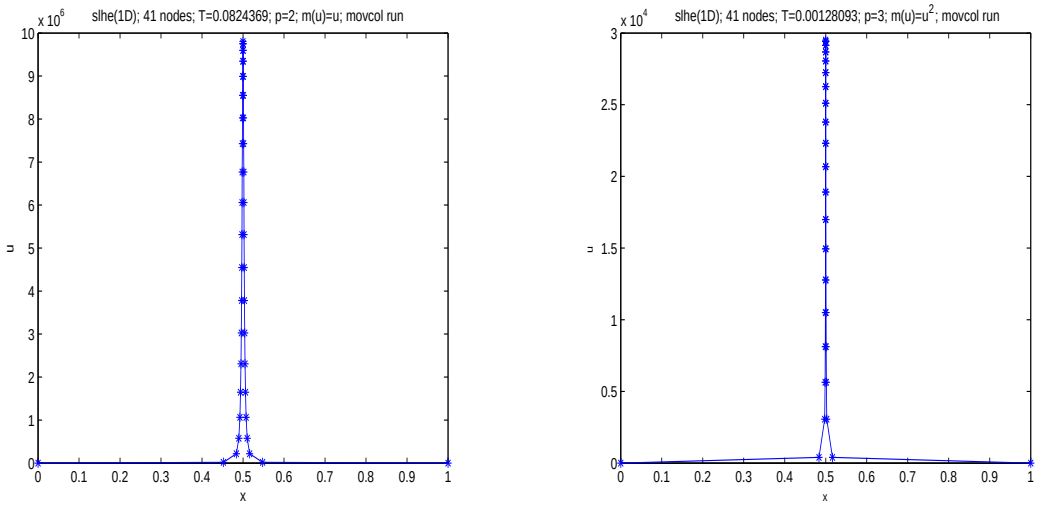

Figure 3.10: Semilinear heat equation (41 node grid): solutions at the time when the algorithm breaks down, for $p = 2$, $m(u) = u$ (left) and $p = 3, m(u) = u^2$ (right) (movcol runs).

also the peak of the initial data, and the results in the Accuracy section (3.4) are showing convergence as the mesh is refined.

The results in the Robustness section (3.6), specifically the increase in nodes to 81 and 161, suggest converging to a value of T, perhaps from above, only limited by tangling problems, though the tangling may be obfuscating the issue. On the tangling issue, the restarted runs in Section 3.6 show a way to overcome this, at least until u_{max} reaches the limit of the computational environment. The comparison with the location-based method "movcol" shows tangling is a major flaw in the BHJx method, but there is no clear evidence in these results as to whether BHJx is more or less efficient than movcol. On choice of monitor, we see the $m(u) = u$ monitor being more robust than $m(u) = u^2$ for the $p = 3$ case. This is not at variance with scale-invariance research suggesting u^{p-1} is the best monitor [20], as that specifically involved the MMPDE method.

Chapter 4

The Area Monitor (PME)

In this chapter we study the porous medium equation in 2D (fully described in Section 1.6.1):-

$$\frac{\partial u}{\partial t} = \nabla \cdot (u^n \nabla u) \ (\mathbf{x} \in \Omega, t > 0), n \text{ being a positive integer; } u \Big|_{t=0} = u_0(\mathbf{x}); u \Big|_{\partial\Omega} = 0. \quad (4.1)$$

We will look at monitor functions of the form $u + a$, where a is a non-negative constant. We refer to this as the *area* monitor, because when a is much larger than u, equidistributing $\int_{\Omega(t)} (u + a) d\Omega$ over N cells effectively means distributing Ω itself into N equal areas. Figure 4.1 shows the interpolated initial solutions on a single mesh, based upon the known similarity solutions with $n = 1$ and $n = 3$ respectively. This mesh has 545 nodes, and is centred on the origin, with a radius of 0.5.

4.1 Applying the BHJx algorithm

The algorithm is applied as in Chapter 2, with the normal (mesh) boundary velocity ($\dot{\xi} \cdot \hat{\mathbf{n}}$) estimated by using the mass monitor (as outlined below). The equations to find the velocity potential, and the mesh velocity from that potential are repeated here for clarity:-

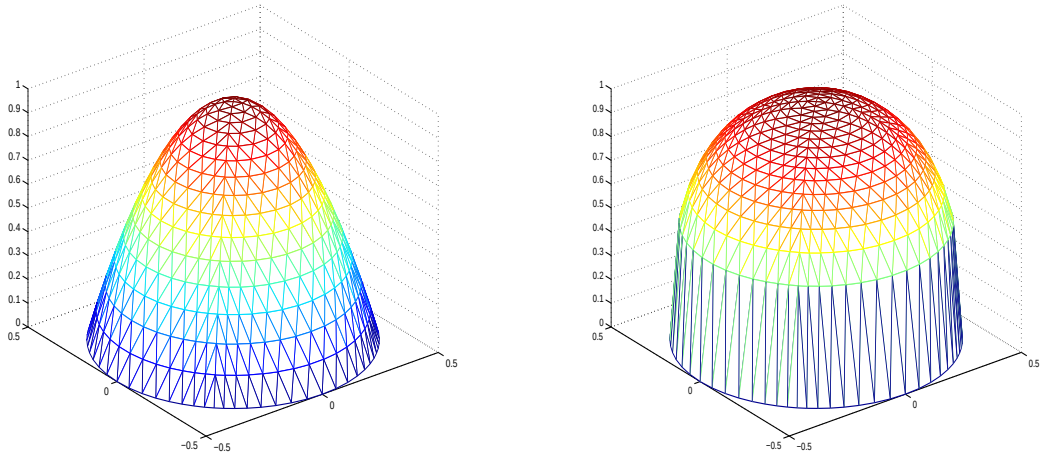

Figure 4.1: Porous medium equation: initial conditions for n=1 (left) and n=3 (right).

$$c_i\dot{\theta}(t)+\int_{\Omega(t)} m(u)\nabla\phi\cdot\nabla w_i d\Omega=\int_{\Omega(t)} w_i m'(u)Lud\Omega+\int_{\partial\Omega(t)} w_i m(u)\dot{\xi}\cdot\hat{\mathbf{n}}dS,\ i=1,2\ldots N. \tag{4.2}$$

$$\dot{\theta}(t)=\int_{\Omega(t)} m'(u)Lud\Omega+\int_{\partial\Omega(t)} m(u)\dot{\xi}\cdot\hat{\mathbf{n}}dS. \tag{4.3}$$

$$\int_{\Omega(t)} w_i\dot{\mathbf{x}}d\Omega=\int_{\Omega(t)} w_i\nabla\phi d\Omega,\quad i=1,2\ldots N, \tag{4.4}$$

with Lu being the PME spatial operator $\nabla\cdot(u^n\nabla u)$. Before solving equations (4.2) and (4.3) in the algorithm, the current values of u and $\mathbf{x}$ are used with the mass monitor ($m(u)=u$) to estimate $\dot{\xi}$ on the boundary in the following steps, which are essentially the core of the mesh velocity calculation in the original BHJ algorithm in Section 1.8:-

A mass total is defined:-

$$\vartheta=\int_{\Omega(t)} ud\Omega. \tag{4.5}$$

Then we calculate the distribution constants:-

$$\gamma_i = \frac{1}{\vartheta(t)} \int_{\Omega(t)} w_i u d\Omega, \quad i = 1,2 \ldots N. \tag{4.6}$$

We calculate the (mass) mesh velocity potential φ from:-

$$\gamma_i \dot{\vartheta} = \int_{\Omega(t)} w_i L u d\Omega + \int_{\Omega(t)} w_i \nabla \cdot (u \nabla \varphi) d\Omega, \quad i = 1,2 \ldots N, \tag{4.7}$$

and

$$\dot{\vartheta} = \int_{\Omega(t)} L u d\Omega + \int_{\Omega(t)} \nabla \cdot (u \nabla \varphi) d\Omega, \tag{4.8}$$

with φ_1 being set to zero to ensure uniqueness. The "mass derivative" $\dot{\vartheta}$ is calculated, but not used. $\dot{\xi}$ is then calculated from the weighted form of $\dot{\xi} = \nabla \varphi$:-

$$\int_{\Omega(t)} w_i \dot{\xi} d\Omega = \int_{\Omega(t)} w_i \nabla \varphi d\Omega, \quad i = 1,2 \ldots N. \tag{4.9}$$

The whole algorithm can be described more simply as:-

BHJx algorithm (with mini-BHJ loop)

Loop:

$u \to (\theta, c_i)$

$u \to (\vartheta, \gamma_i)$

$(u, \gamma_i) \to \varphi.$

$\varphi \to \dot{\xi}.$

$(u, c_i, \dot{\xi}) \to \phi.$

$\phi \to \dot{\mathbf{x}}.$

$(u, \dot{\mathbf{x}}) \to \dot{u}.$

Update **x** and u.

End Loop.

4.2 Accuracy and mesh control

As the porous medium equation has a known similarity solution, we can estimate accuracy of BHJx by comparison against this case. Orders of convergence for the solution and mesh are shown in Figure 4.2, for $n = 1$ and $n = 3$, for meshes of 545, 2113 and 8321 nodes (so dx decreasing by 50%), and a timestep of 0.0001 for 545 nodes (and then reducing by 25%), for different values of a in the area monitor $m(u) = u + a$. These runs were done to T=0.1: two of the final meshes are shown in Figure 4.3, and the known solution at that time for the mesh positions in Figure 4.3, for $n = 1$ and $n = 3$, are shown in Figure

4.4, where we can see close agreement, qualitatively. The values for $a = 0$ in the log-log graphs correspond to the foundation established in the BHJx development (Section 2.2), and are also comparable to the original BHJ study, using the mass monitor [4].

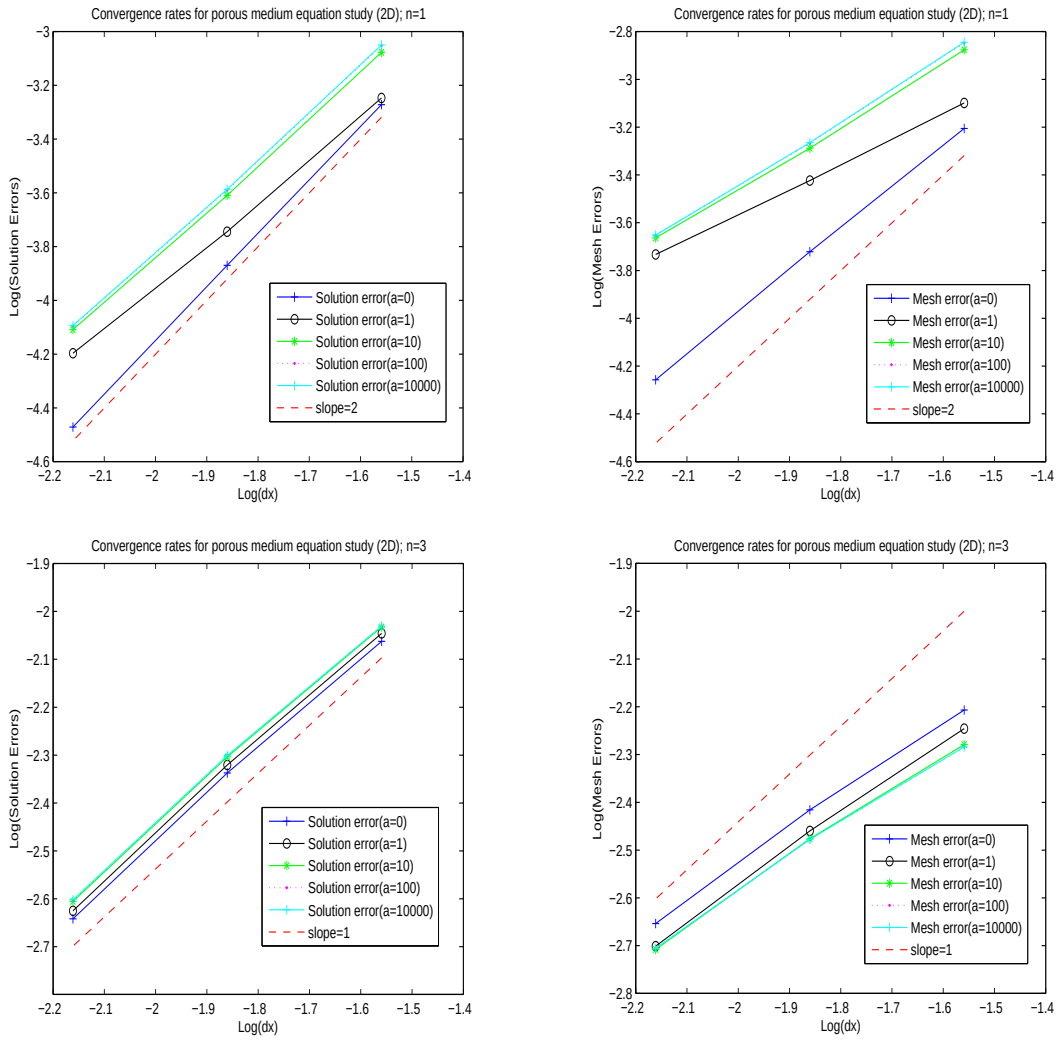

Figure 4.2: Porous medium equation, area monitors. Orders of convergence for: L^2 mesh error, n=1 (top left), L^2 boundary error, n=1 (top right), L^2 mesh error, n=3 (bottom left), L^2 boundary error, n=3 (bottom right).

For the solution errors, for both $n = 1$ and $n = 3$, we can see the values are slowly increasing as a increases, with the rate of this increase lessening as a gets beyond 10. The order of accuracy for the solution error remains at 2 for $n = 1$, and 1 for $n = 3$ though, apart from the one anomaly of $u + 1$ for the $n = 1$ case. A similar result is seen for the mesh

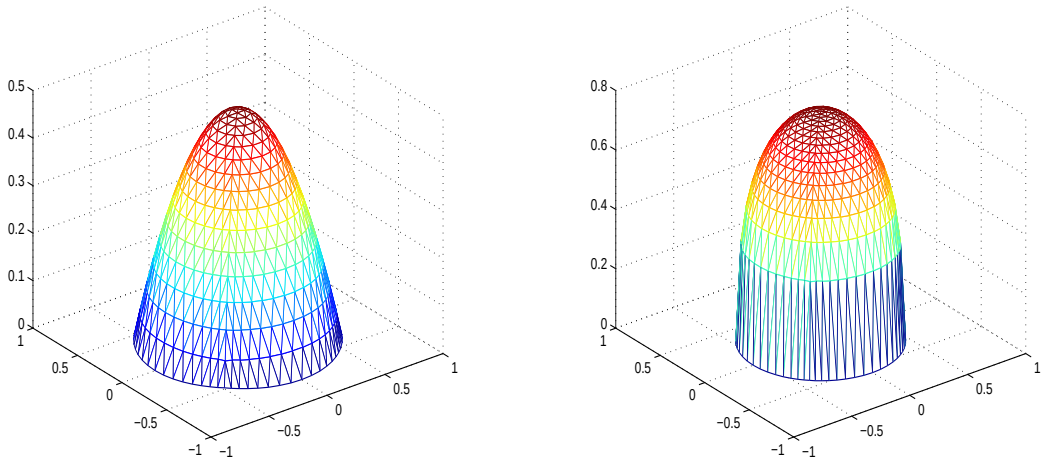

Figure 4.3: Porous medium equation: meshes at T=0.1, for monitor $u + 10000$, 545-node mesh, $n = 1$ (left) and n=3 (right), with approximation replaced by known solution.

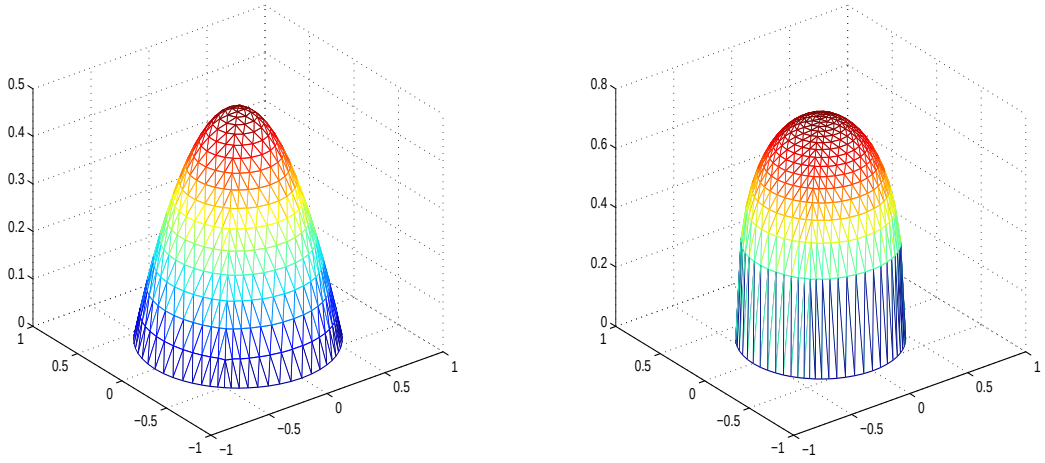

Figure 4.4: Porous medium equation: meshes at T=0.1, for monitor $u + 10000$, 545-node mesh, $n = 1$ (left) and n=3 (right).

error for $n = 1$, with $u + 1$ being anomalous again. The mesh errors for $n = 3$ are actually decreasing in value, but the order of accuracy is also decreasing here. For $a = 100$ and $a = 10000$, the order of convergence (on the two most refined meshes) for the mesh error in the $n = 3$ case was $3/4$ for both values - the slopes here coincide (bottom right graph of Figure 4.2).

At a fundamental level, the algorithm works and shows the area monitor has a poorer order of convergence than the mass monitor, with the error values getting higher as a increases. We look more closely now at the action of the area monitor. We expect it to be equidistributing area if the initial area is equidistributed, and a further case was run to evaluate this, with $a = 10^6$. This was done on the 545-node mesh, for n=3, but with n=1 initial conditions[1]. In this case, the slope at the boundary steepens as the run progresses and there is therefore more mass per cell near the boundary. With the mass monitor, we would expect cells near the boundary to decrease in area, to maintain the mass/cell ratio relative to the whole domain. With the area monitor, we expect these cells to retain their initial area distribution, in this case an equidistribution. The graphs in Figure 4.5 (initial mesh and zoom-in), Figure 4.6 (meshes at T=10 for mass and area monitors) and Figure 4.7 (zoom-in of graphs in Figure 4.6) confirm this result. This does leave the question of whether the monitor distribution was maintained during *all* of the run - this question will be addressed in Chapter 5.

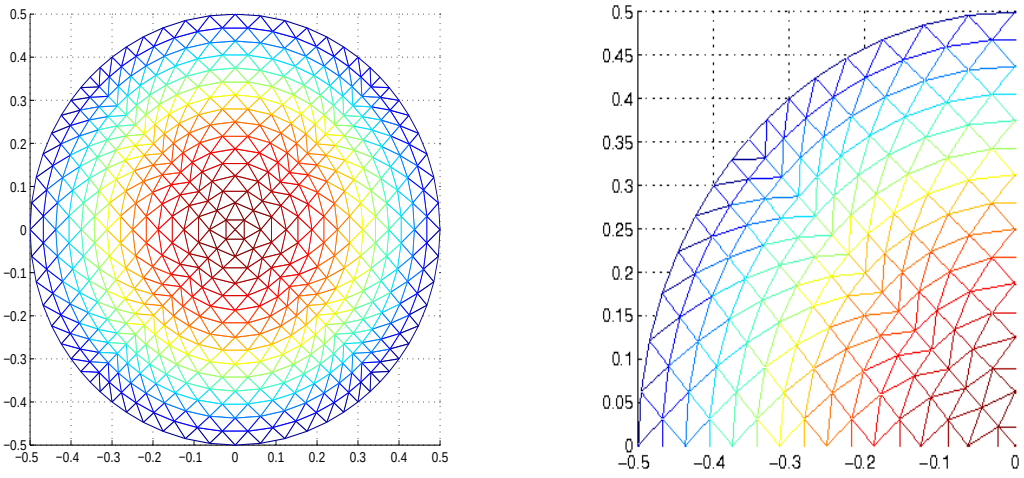

Figure 4.5: Porous medium equation, monitor comparison: initial mesh and zoom-in.

[1]In the accuracy study in Section 4.2, all the errors for $a = 10^6$ were equal to those for $a = 10^4$, to two decimal places.

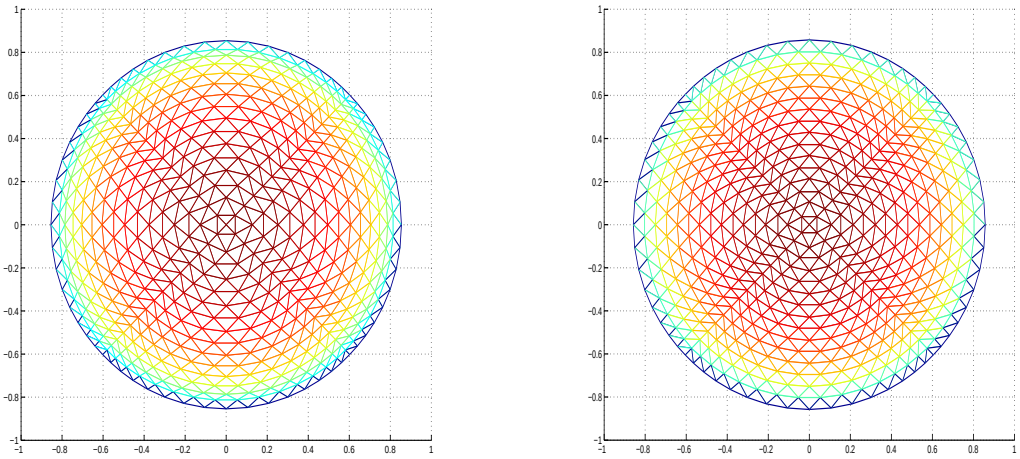

Figure 4.6: Porous medium equation, monitor comparison: mesh planviews at T=10 for mass monitor (left) and area monitor (a=1000000) (right).

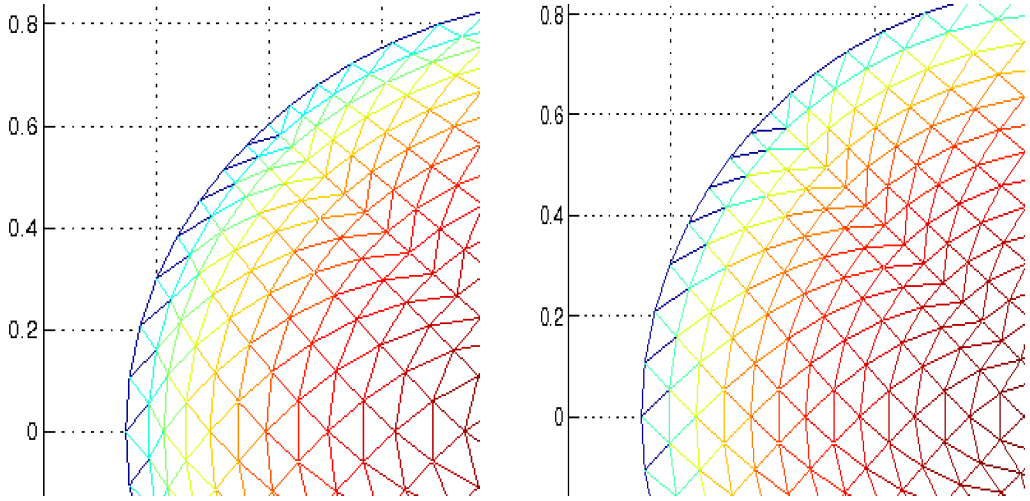

Figure 4.7: Porous medium equation, monitor comparison: zoom of mesh planviews at T=10 for mass monitor (left) and area monitor (a=1000000) (right).

4.3 Robustness

The last run in Section 4.2 shows some robustness, for a non-similarity solution case, as an $n = 3$ problem was run with $n = 1$ initial conditions, and completed through to T=10.0. In addition to this, Figure 4.8 shows the result for a larger mesh, of 33025 nodes and initial radius 0.5, running to T=0.01 (this being as long as computational resources would allow), for both $n = 1$ and $n = 3$, this time with matching (in n) initial conditions. The

monitor was $u + 100$ in this case.

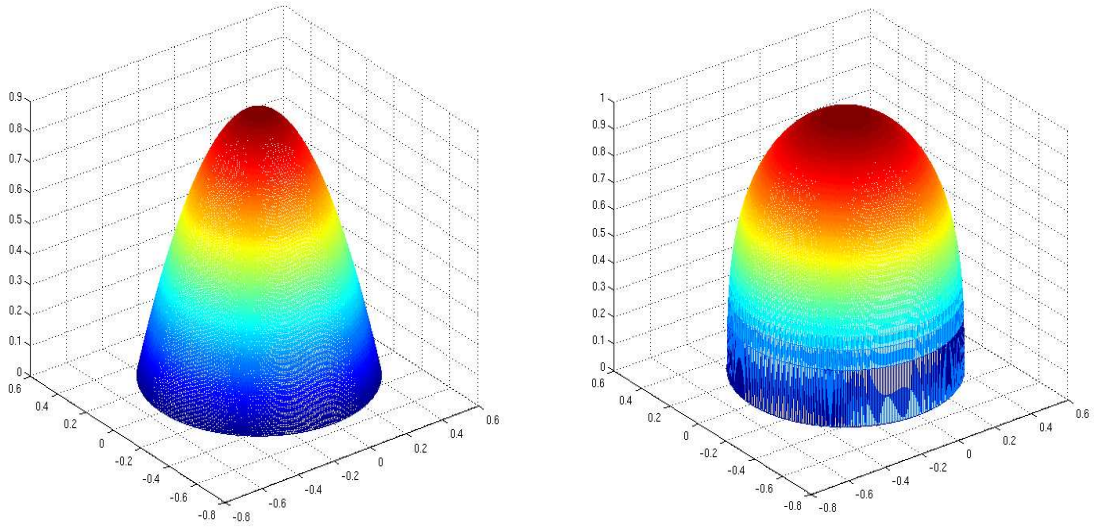

Figure 4.8: Robustness test: area monitor final mesh for 33025 nodes, T=0.01, n=1 (left) and n=3 (right). For a=100.

As a further robustness test, both mass and area monitors have been run on an initial mesh that does not exhibit radial symmetry. The initial mesh is shown in Figure 4.9 and the meshes at $T = 0.01$, $T = 0.02$ and $T = 0.1$ for both monitors are shown in Figures 4.10, 4.11 and 4.12 respectively, running with $n = 1$. Although there is a slight difference between the meshes for the two monitors at these times, we can see the two initial peaks merge as the meshes evolve for both monitors, and the solution tending towards a radially symmetric similarity solution.

With the same "twin-peak" initial conditions, the meshes for $n = 2$ at $T = 1.0$ are shown for the mass and area monitors in Figures 4.13 and 4.14. In this case the area monitor clearly preserves the shape of the initial mesh better than the mass monitor. This is particularly obvious at the centre of the domain where the cells are twisted by the mass monitor.

As a final robustness test, both mass and area monitors have been run, with $n = 1$, for two domains where the initial mesh for n=1 (545 nodes, radius = 0.5), shown in Figure 4.1 (left) has had its mesh positions sinusoidally perturbed (but each node keeping its value

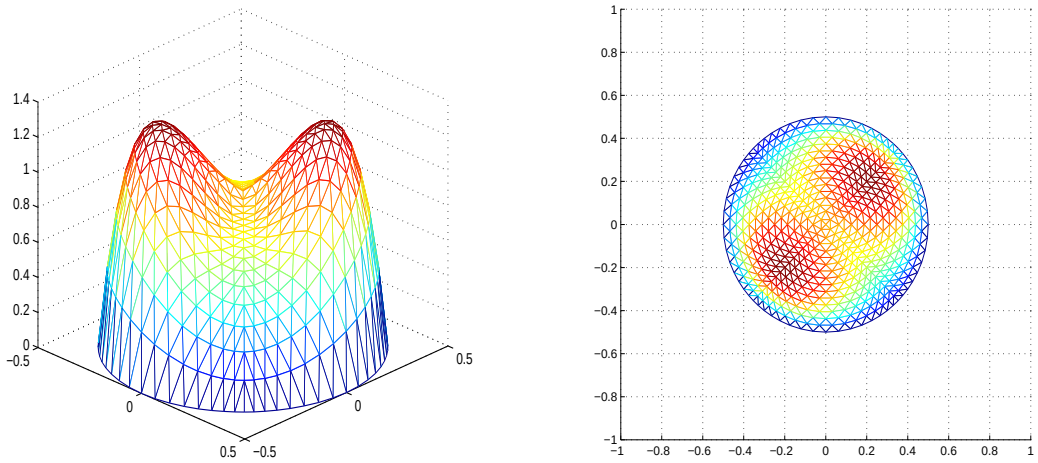

Figure 4.9: Robustness test: Non (radially) symmetric initial conditions with twin peaks. Initial mesh (n=1, 545 nodes).

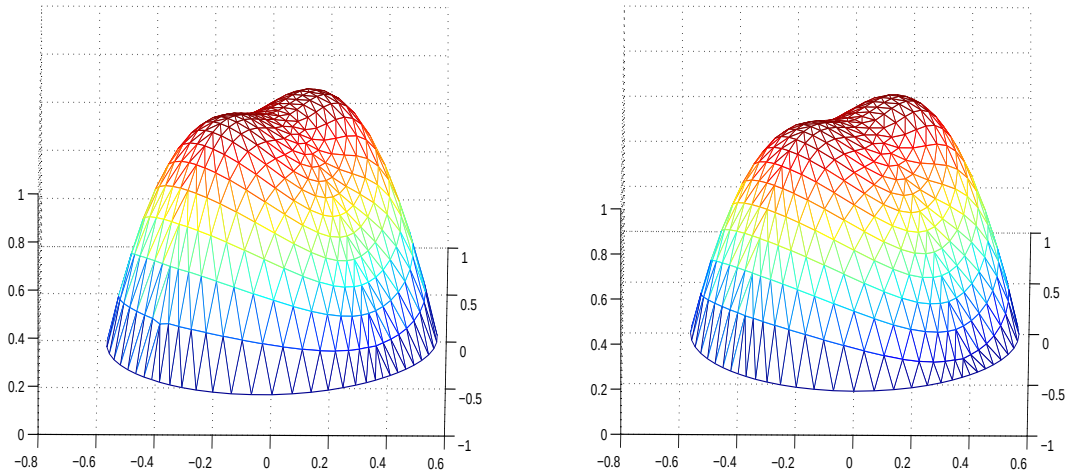

Figure 4.10: Robustness test: Twin peak meshes at T=0.01, n=1. Mass monitor on left, area monitor on right.

of u), to form the mesh in Figure 4.15, with its plan view shown on the right. For both the mass and area monitor, the timestep was reduced by 1/100, as the runs tangled very quickly with the usual timestep[2]of 0.0001.

[2]Following this analysis, these runs were repeated with just dt/10, and the final mesh positions and values of u were equal to the dt/100 case, to four decimal places.

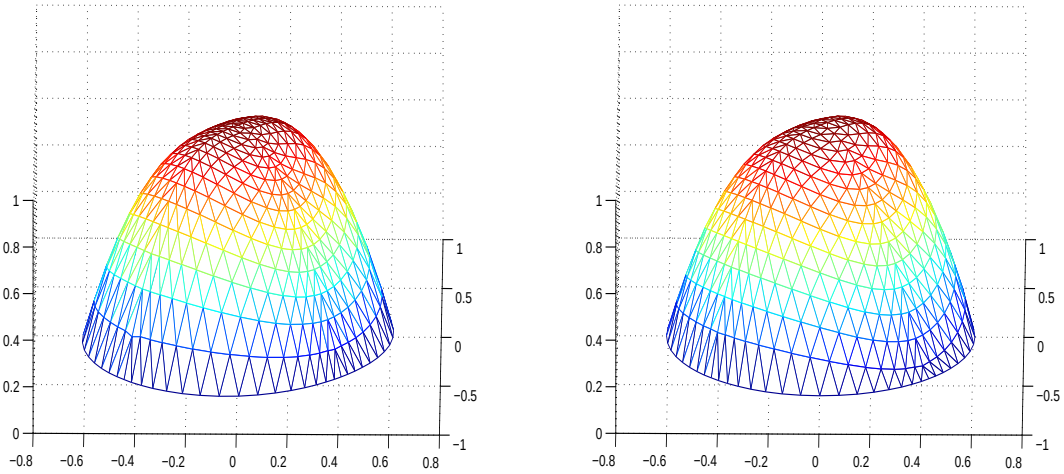

Figure 4.11: Robustness test: Twin peak meshes at T=0.02, n=1. Mass monitor on left, area monitor on right.

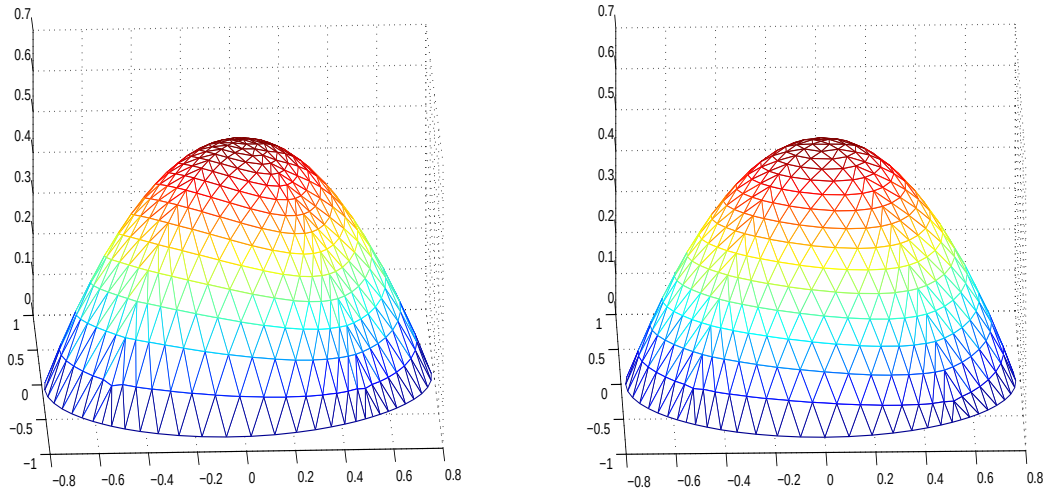

Figure 4.12: Robustness test: Twin peak meshes at T=0.1, n=1. Mass monitor on left, area monitor on right.

For these initial conditions, the mass monitor ran to T=0.008 before tangling, and the mesh just before the tangling is shown in Figure 4.16 , with two zooms near the element about to tangle. In the closer zoom (bottom), the elements adjacent to the circled node may seem to be visibly pristine. However, when we look at the same point in the mesh, at the same time, for the area monitor (with $a = 1 \times 10^6$) in Figure 4.17, we can see that the

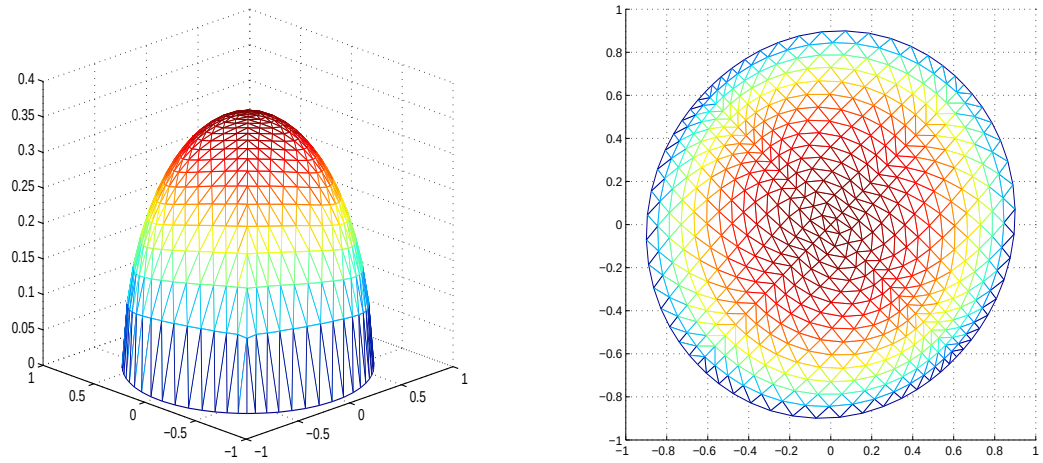

Figure 4.13: Robustness test: Twin peak mesh and planview at T=1.0, n=2, mass monitor.

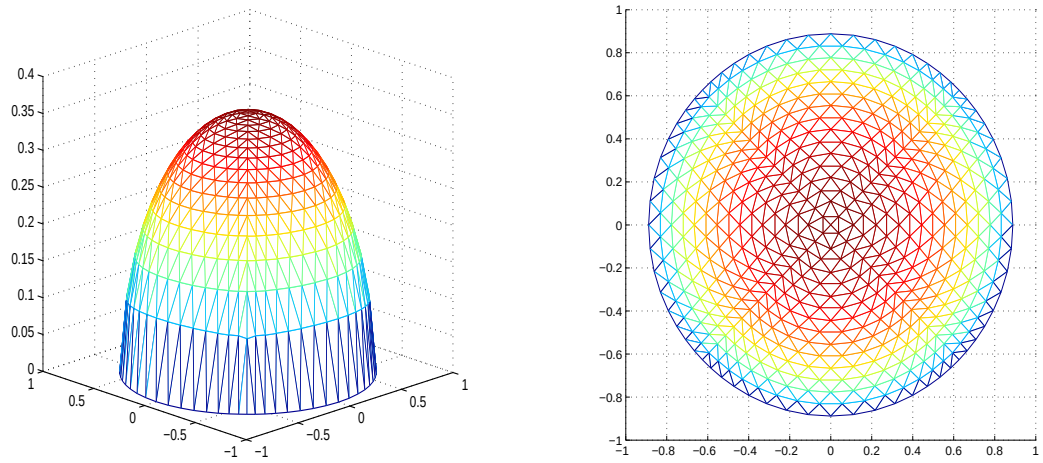

Figure 4.14: Robustness test: Twin peak mesh and planview at T=1.0, n=2, area monitor.

same node should have six lines emanating from it, but with the mass monitor, there are only five.

The mesh (and zoom) for the mass monitor at an earlier time of T=0.0075 are shown in Figure 4.18, and now we can see that that “5-node” is actually a 6-node with two edges almost adjacent - at the time of tangling (one timestep after T=0.008), these then overlap. Hence the area monitor has proved more robust than the mass monitor - it went further, to

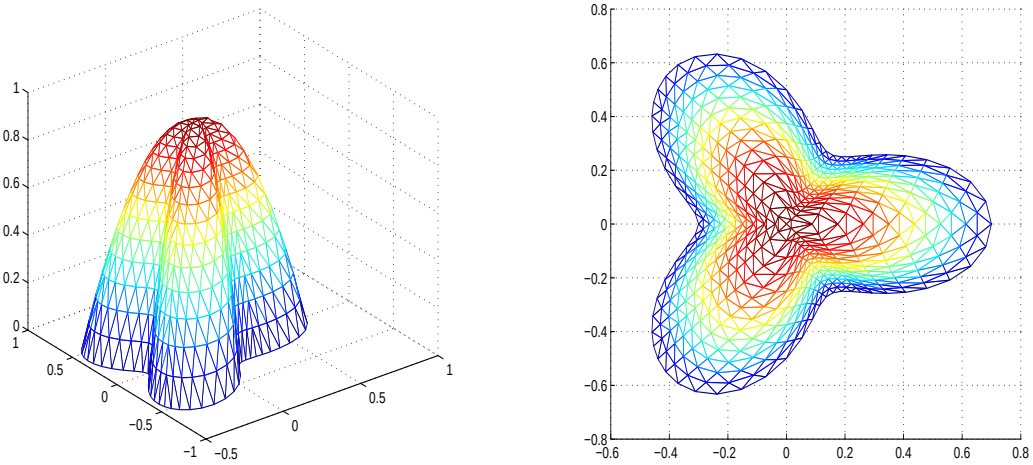

Figure 4.15: Robustness test: initial mesh (n=1, 545 nodes, radius = 0.5) sinusoidally perturbed (1).

T=0.0125, before tangling. The mesh just before tangling, for the area monitor, is shown in Figure 4.19, where we can see the same element is about to collapse.

Secondly, the above run was repeated, but with the sinusoidal perturbation smaller than the mesh shown in Figure 4.15, to see if the area monitor would run right through to a circular domain, whereas the mass monitor would fail early on. For these runs, the timestep was reduced by 1/10 to dt=0.00001, from the usual timestep of 0.0001 The initial mesh for these runs and its planview are shown in Figure 4.20. The mass monitor tangled at T=0.232, and the mesh just before the tangle, with a planview and zoom, are shown in Figure 4.21. The area monitor tangled at T=1.299, and the mesh just before that run tangled, with a planview and zoom, are shown in Figure 4.22. As with the larger perturbation, the area monitor has got further than the mass monitor before tangling, in this case, more than five times as far, and the mesh has proceeded through to an almost circular domain, though it has ultimately tangled. The zooms of the planviews are at the farthest point on the left of the x-axis (on the line y=0). These show a pattern slightly different from the first case, but still we see elements are about to have their edges cross over, so that their area becomes negative.

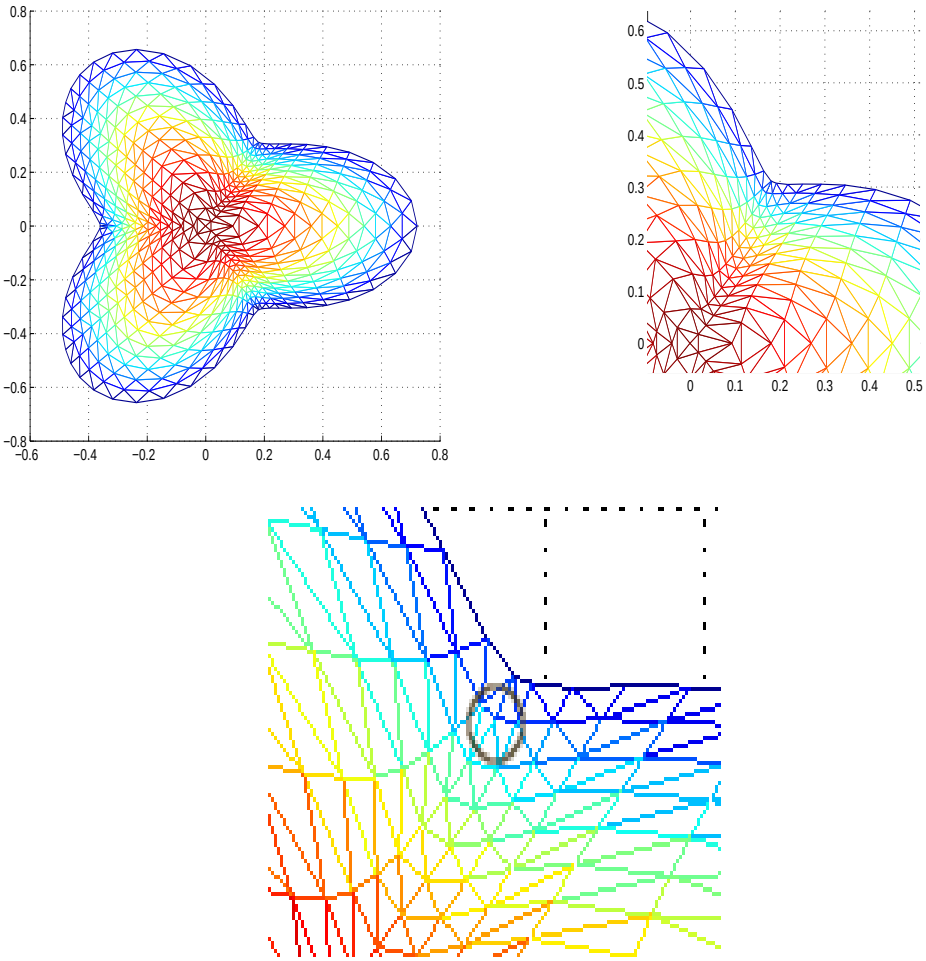

Figure 4.16: Robustness test: perturbed mesh (1) at T=0.008, for mass monitor.

4.4 Summary and Discussion

The assessment has shown that the BHJx method applied to the porous medium equation in 2D, with area monitor $(u + a)$, and prescribed normal boundary velocity, essentially works, and this for different values of a. When it has been compared to the mass monitor, we see a poorer order of convergence as meshes are refined, and this is generally worsened as a increases. Looking at the differences in monitors in more detail, we can see that both monitors are actually doing what is expected of them - attempting to conserving their monitor distribution. For the area monitor, this behaviour might seem to be a disadvantage, as we have less accuracy. However, when we have analysed meshes which

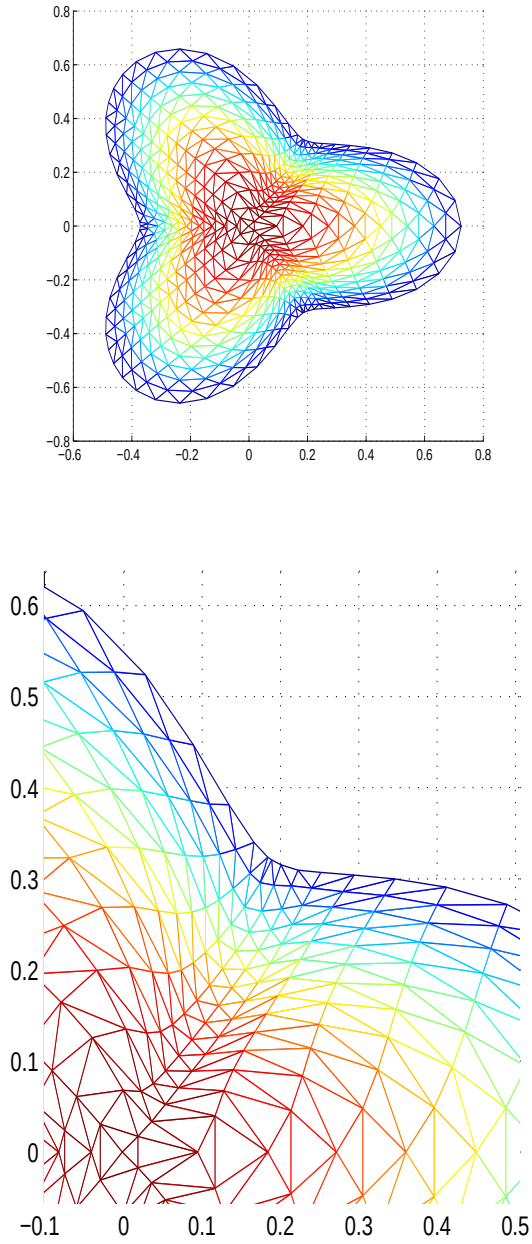

Figure 4.17: Robustness test: perturbed mesh (1) at T=0.008, for area monitor ($a = 1 \times 10^6$).

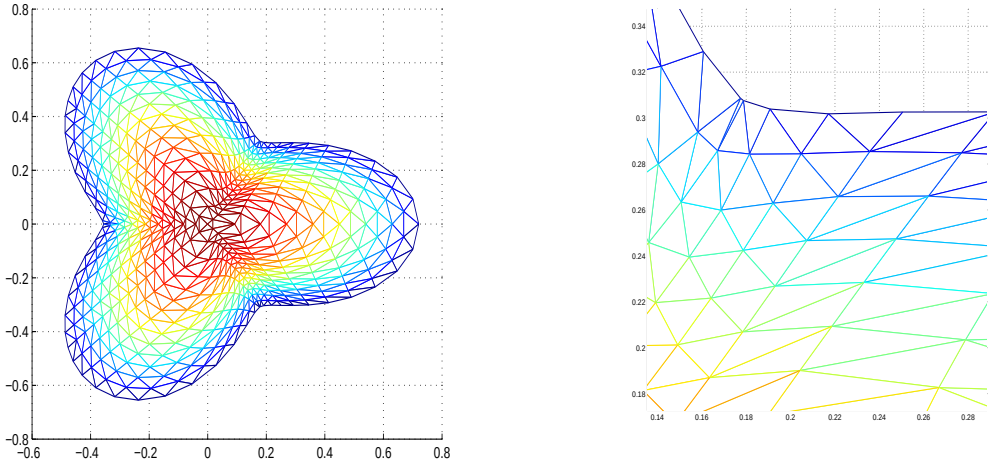

Figure 4.18: Robustness test: perturbed mesh (1) at T=0.0075, for mass monitor.

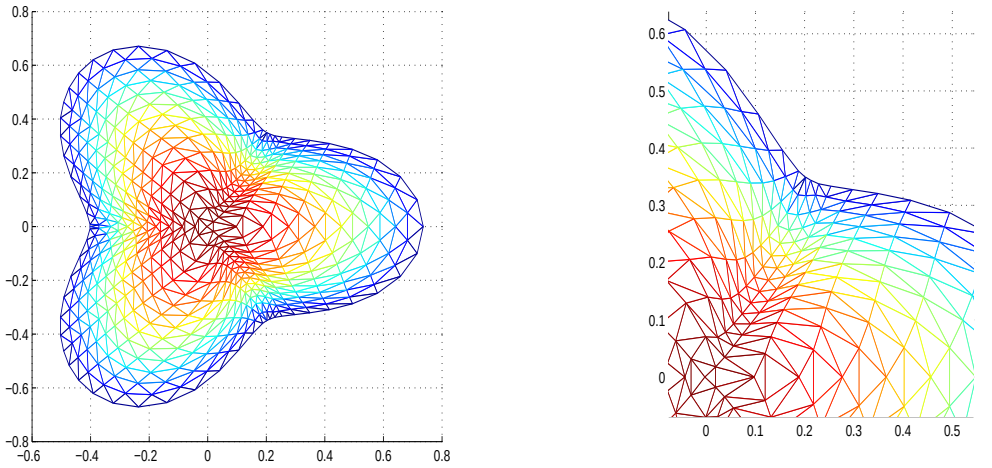

Figure 4.19: Robustness test: perturbed mesh (1) at T=0.0125, for area monitor ($a = 1 \times 10^6$).

are not (radially) symmetric, we can see the area monitor has an advantage, in terms of robustness.

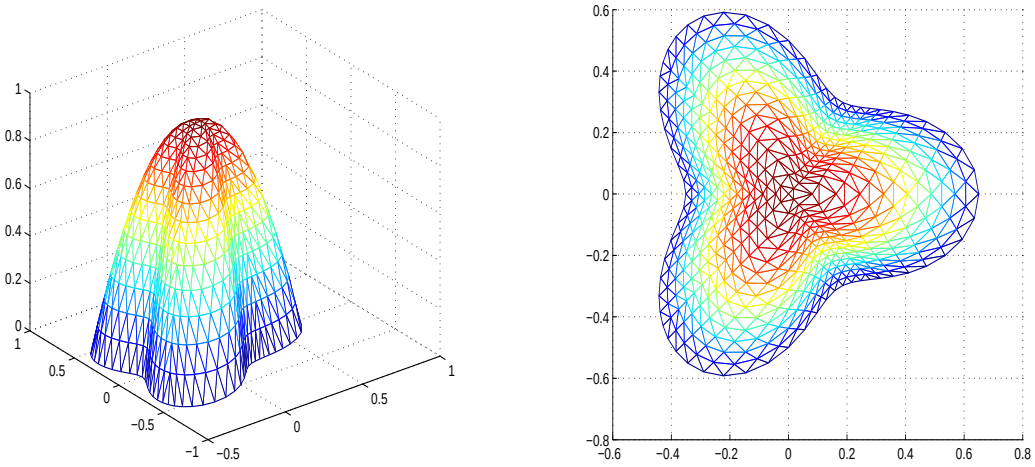

Figure 4.20: Robustness test: initial mesh (n=1, 545 nodes, radius = 0.5) sinusoidally perturbed (2).

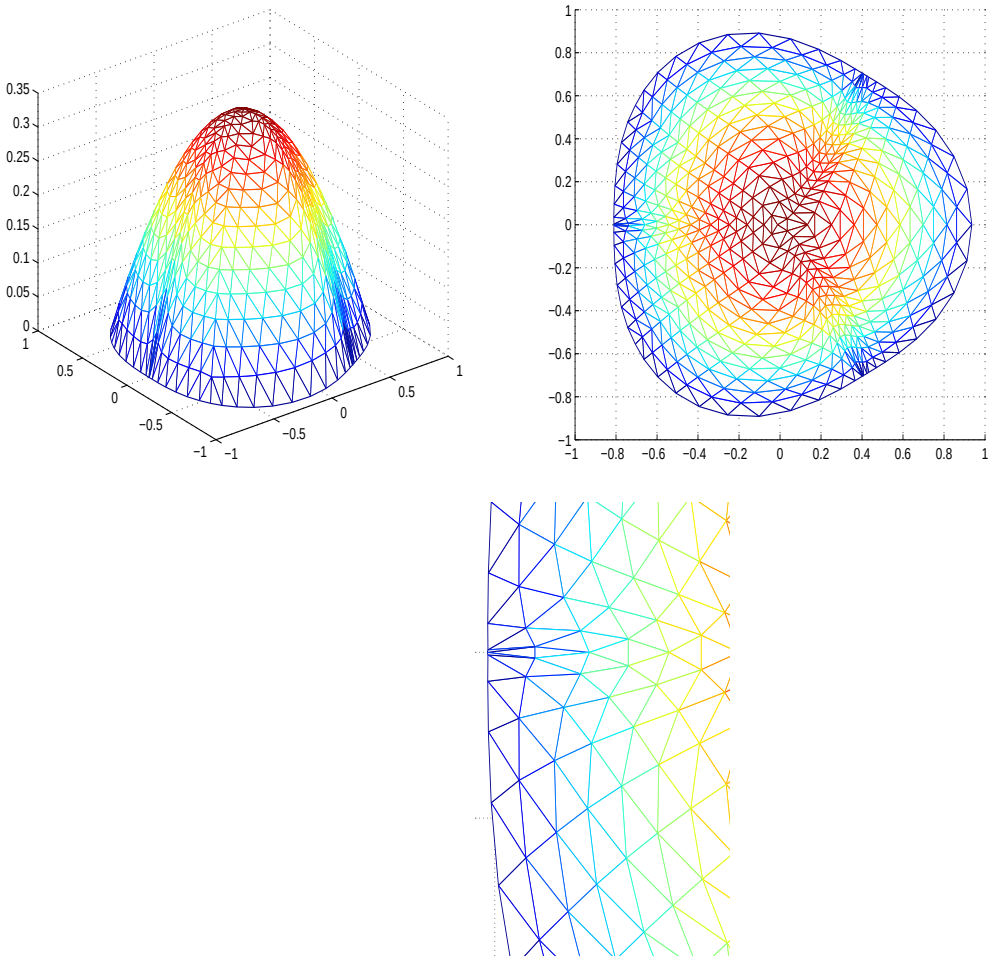

Figure 4.21: Robustness test: perturbed mesh (2) at T=0.232, for mass monitor.

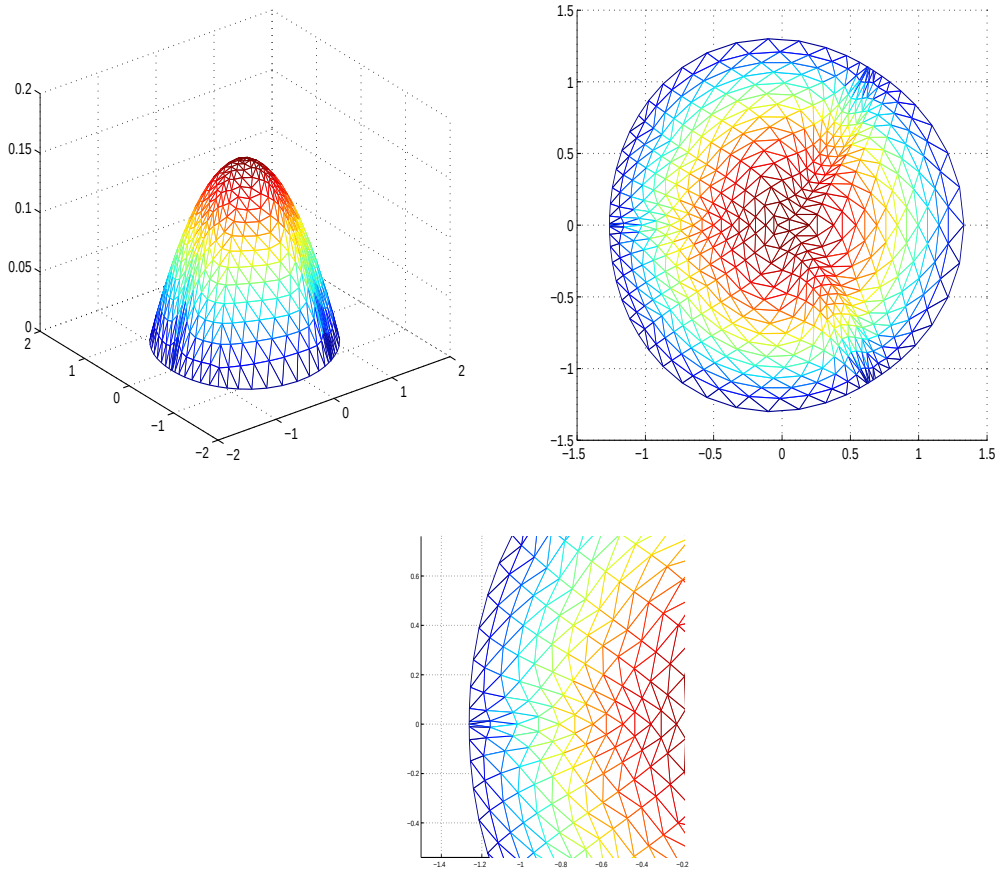

Figure 4.22: Robustness test: perturbed mesh (2) at T=1.299, for area monitor ($a = 1 \times 10^6$).

Chapter 5

The Arc-length Monitor (PME)

5.1 Background

In this chapter we study the performance of the BHJx algorithm, using a monitor that we refer to as the arc-length monitor, applied to the porous medium equation. The bulk of the work here is in 1D, but there is a short 2D study at the end of the chapter. This PDE is fully described in Section 1.6.1. In 1D, it has the form:-

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}(u^n \frac{\partial u}{\partial x}) \ (x \in [a(t), b(t)], t > 0), n \text{ being a positive integer;}$$

$$u\Big|_{t=0} = u_0(x); u\Big|_{x=a(t)} = 0; u\Big|_{x=b(t)} = 0. \tag{5.1}$$

As the PME has a known analytical solution, we will look at convergence to that solution, as well as robustness. We consider the effect of optimising initial conditions, so that the initial mesh is equidistributed for the arc-length monitor. We also look beyond the derivation of the mesh velocity purely as an algorithmic device - we will see if we can actually maintain the initial distribution of the monitor function $m(u)$ - which amounts

to keeping the c_i in equation (2.6) constant throughout the time domain. In fact, we will look at a way of recovering u that forces the values of c_i to be maintained, whilst still robustly modelling the porous medium equation. This second method will be referred to in the sequel as "ALE+" as it takes the nodal values from the usual ALE equation (2.4) as a starting point, and applies an extra step to them. The first method, i.e, using the nodal values as they are, will be referred to as "ALE". Hence the effects of moving away from an equidistributed mesh will be quantified here.

5.2 Applying the BHJx algorithm

The algorithm is applied as in Chapter 2, with the normal (mesh) boundary velocity estimated by using the mass monitor. The equations to find the velocity potential, and the mesh velocity from that potential are repeated here for clarity:-

$$c_i\dot{\theta}(t)+\int_{\Omega(t)} m(u)\nabla\phi\cdot\nabla w_i d\Omega=\int_{\Omega(t)} w_i m'(u)Lud\Omega+\int_{\partial\Omega(t)} w_i m(u)\dot{\xi}\cdot\hat{\mathbf{n}}dS,\ i=1,2\ldots N. \tag{5.2}$$

$$\dot{\theta}(t)=\int_{\Omega(t)} m'(u)Lud\Omega+\int_{\partial\Omega(t)} m(u)\dot{\xi}\cdot\hat{\mathbf{n}}dS. \tag{5.3}$$

$$\int_{\Omega(t)} w_i\dot{\mathbf{x}}d\Omega=\int_{\Omega(t)} w_i\nabla\phi d\Omega,\quad i=1,2\ldots N, \tag{5.4}$$

with Lu being the PME spatial operator $\nabla\cdot(u^n\nabla u)$. As in Chapter 4, we use the core of the mesh velocity calculation in the original BHJ algorithm in Section 1.8 to estimate $\dot{\xi}$ from current values of u and $\mathbf{x}$. Specifically, before equation (5.2) in the algorithm, the mass monitor ($m(u)=u$) is used to provide the normal mesh velocity $\dot{\xi}$ as follows:-

A mass total is defined:-

$$\vartheta = \int_{\Omega(t)} u d\Omega. \tag{5.5}$$

Then we calculate the distribution constants:-

$$\gamma_i = \frac{1}{\vartheta(t)} \int_{\Omega(t)} w_i u d\Omega, \quad i = 1,2 \ldots N. \tag{5.6}$$

We calculate the (mass) mesh velocity potential φ from:-

$$\gamma_i \dot{\vartheta} = \int_{\Omega(t)} w_i L u d\Omega + \int_{\Omega(t)} w_i \nabla \cdot (u \nabla \varphi) d\Omega, \quad i = 1,2 \ldots N, \tag{5.7}$$

and

$$\dot{\vartheta} = \int_{\Omega(t)} L u d\Omega + \int_{\Omega(t)} \nabla \cdot (u \nabla \varphi) d\Omega, \tag{5.8}$$

with φ_1 being set to zero to ensure uniqueness. The "mass derivative" $\dot{\vartheta}$ is calculated, but not used.

$\dot{\xi}$ is then calculated from the weighted form of $\dot{\xi} = \nabla \varphi$:-

$$\int_{\Omega(t)} w_i \dot{\xi} d\Omega = \int_{\Omega(t)} w_i \nabla \varphi d\Omega, \quad i = 1,2 \ldots N. \tag{5.9}$$

In 1D, our domain $\Omega(t)$ becomes a moving interval $[a(t), b(t)]$ and equations (5.2), (5.3) and (5.4) become:-

$$c_i \dot{\theta}(t) + \int_{a(t)}^{b(t)} m(u) \frac{\partial \phi}{\partial x} \frac{\partial w_i}{\partial x} dx = \int_{a(t)}^{b(t)} w_i m'(u) L u \, dx + \left[w_i m(u) \dot{\xi} \right]_{a(t)}^{b(t)}, \; i = 1,2 \ldots N, \tag{5.10}$$

$$\dot{\theta}(t) = \int_{a(t)}^{b(t)} m'(u) L u \, dx + \left[m(u) \dot{\xi} \right]_{a(t)}^{b(t)}, \; i = 1,2 \ldots N, \tag{5.11}$$

and:-

$$\int_{a(t)}^{b(t)} w_i \dot{x} dx = \int_{a(t)}^{b(t)} w_i \frac{\partial \phi}{\partial x} dx, \quad i = 1, 2 \ldots N. \tag{5.12}$$

The monitor function for arc-length in 1D is:-

$$m(\frac{\partial u}{\partial x}) = \sqrt{(}1 + (\frac{\partial u}{\partial x})^2). \tag{5.13}$$

Note that this requires an amendment to equation (2.8) (and so also equations (5.10) and (5.11)) since they assumed m was a function of u only. Specifically, we now deal with the time derivative of $m(\frac{\partial u}{\partial x})$ as follows:-

$$\begin{aligned} \frac{\partial}{\partial t}\left(m\left(\frac{\partial u}{\partial x}\right)\right) &= m'\left(\frac{\partial u}{\partial x}\right)\frac{\partial}{\partial t}\left(\frac{\partial u}{\partial x}\right) \\ &= m'\left(\frac{\partial u}{\partial x}\right)\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial t}\right) \\ &= m'\left(\frac{\partial u}{\partial x}\right)\frac{\partial}{\partial x}(Lu). \end{aligned} \tag{5.14}$$

Then writing $v = \frac{\partial u}{\partial x}$ (so for the arc-length monitor, $m(v) = \sqrt{(}1 + v^2)$), equations (5.10) and (5.11) (our moving mesh driver) become:-

$$c_i \dot{\theta}(t) + \int_{a(t)}^{b(t)} m(v) \frac{\partial \phi}{\partial x} \frac{\partial w_i}{\partial x} dx = \int_{a(t)}^{b(t)} w_i m'(v) \frac{\partial}{\partial x} Lu \, dx + \left[w_i m(v) \dot{\xi} \right]_{a(t)}^{b(t)}, i = 1, 2 \ldots N, \tag{5.15}$$

and

$$\dot{\theta}(t) = \int_{a(t)}^{b(t)} m'(v) \frac{\partial}{\partial x} Lu \, dx + \left[m(v) \dot{\xi} \right]_{a(t)}^{b(t)}, \; i = 1, 2 \ldots N. \tag{5.16}$$

With piecewise linear elements, v is constant on each element, so the terms $m(v)$ and $m'(v)$ are actually simpler to calculate than when we have $m(u)$ terms. However, as L is

a second order operator, we now have some third derivative terms in (5.15) and (5.16) which we need to treat.

We can approximate Lu by letting $q = Lu$ and then manipulating the weak form of that identity [5]:-

$$\int_{a(t)}^{b(t)} w_i q dx = \int_{a(t)}^{b(t)} w_i Lu\, dx,\ i = 1,2\ldots N. \tag{5.17}$$

For the PME this becomes:-

$$\int_{a(t)}^{b(t)} w_i q\, dx = \int_{a(t)}^{b(t)} w_i \frac{\partial}{\partial x}(u^n \frac{\partial u}{\partial x})\, dx,\ \ i = 1,2\ldots N. \tag{5.18}$$

As $u = 0$ on the boundary, integrating by parts gives:

$$\int_{a(t)}^{b(t)} w_i q\, dx = -\int_{a(t)}^{b(t)} \frac{\partial w_i}{\partial x}(u^n \frac{\partial u}{\partial x})\, dx,\ \ i = 1,2\ldots N, \tag{5.19}$$

which can be solved to find q, and so approximate $\frac{\partial q(=Lu)}{\partial x}$ in equations (5.15) and (5.16). This method of finding Lu will be used in most of the following work, but we will also look at one variation, where we use cubic splines [68] to evaluate Lu, using only the nodal values of u.

The whole algorithm can be described more simply as:-

BHJx algorithm (with mini-BHJ loop and Lu estimation)

Loop:

$u \rightarrow (\theta, c_i)$

$u \rightarrow (\vartheta, \gamma_i)$

$(u, \gamma_i) \rightarrow \varphi.$

$\varphi \rightarrow \dot{\xi}.$

$u \rightarrow (q_i)$

$(u, c_i, \dot{\xi}, q_i) \rightarrow \phi.$

$\phi \rightarrow \dot{\mathbf{x}}$.

$(u, \dot{\mathbf{x}}) \rightarrow \dot{u}$.

Update $\mathbf{x}$ and u.

End Loop.

5.3 ALE results

5.3.1 Basic Cases

The initial conditions for this chapter have always been derived from the known similarity solution [4], using equation (1.26). The following cases have all been run with the parameter n set to 1 - cases for $n = 3$ are considered only in Section 5.5. In the first set of cases, we sub-divide the interval $[-0.5, 0.5]$ into 10, 20 and 40 equally-spaced intervals (and so there are 11, 21 and 41 nodes), and ran the algorithm for a *run* time of $T = 1.0$. With an initial radius of 0.5, the start time of these runs is $t_0 = 0.041667$, from equation (1.27). These initial conditions and the grids at the end of the three runs are shown in Figure 5.1, together with the known solution for comparison.

We can see a convergence (in Figure 5.1) of the approximation to the known solution at the internal nodes, but it is not clear if there is any convergence at the boundary and near-boundary nodes. This is quantified in Figure 5.2, where we also show the evolution of the monitor distribution constants c_i. In these evolution plots, the legend refers to the values of c_i at three points in the first half of the interval, so for 21 nodes for example, the boundary is Node 1, the "quarter-point" is Node 6 and the origin is Node 11. For 11 nodes, the quarter-point is taken as Node 3. The boundary error is the absolute difference between the approximated and the known boundary node positions. We can see a convergence order that appears to be slightly higher than $\frac{1}{2}$ for the L^2 error, and we can see there is a convergence for the boundary error, with an order slightly higher than $\frac{1}{4}$, which are smaller orders than those obtained with the mass monitor [4]. Concerning the distribution

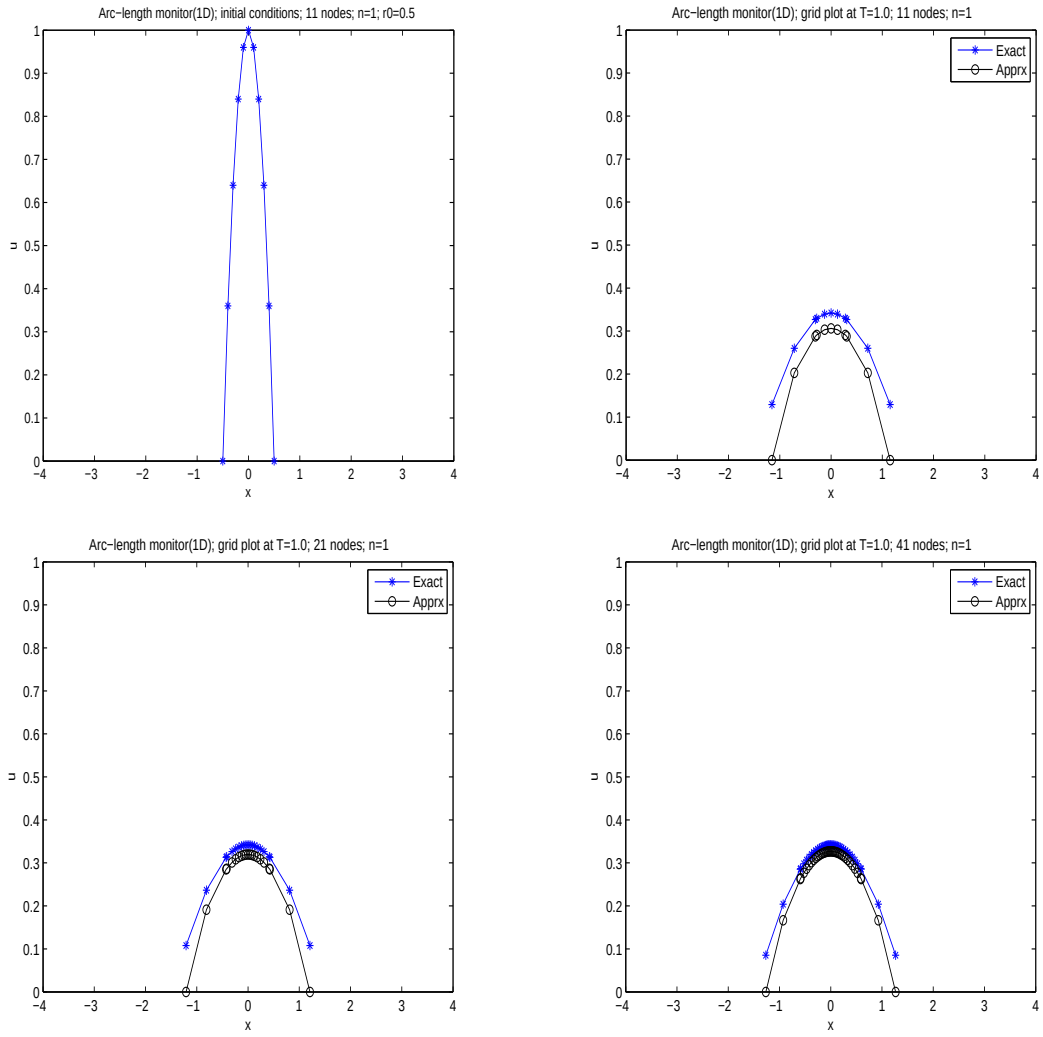

Figure 5.1: Grid evolution for ALE runs with unoptimised initial data (grid uniformly spaced), 11, 21 and 41 nodes. Graphs show initial grid (top left), and exact (known) solution and approximation at T=1.0.

constants (c_i), we can see they are not staying constant at first, though there is some settling down later in the runs. Recall that, although maintaining a constant distribution of the monitor is the driver for the node movement algorithm, there is nothing in the final scheme that absolutely forces this (but see Section 5.4 below).

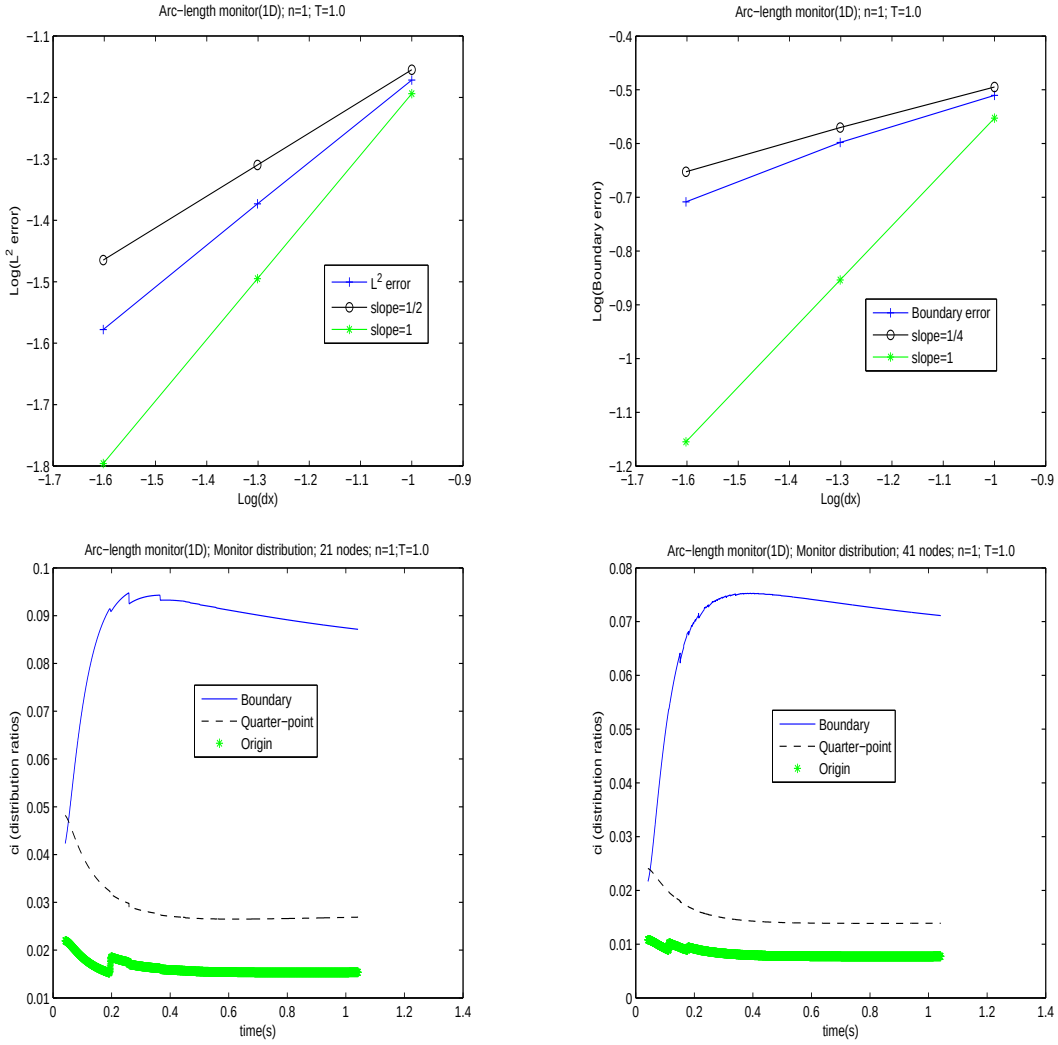

Figure 5.2: Convergence rates (solution and boundary error) and monitor distribution evolution for ALE runs with unoptimised initial data.

5.3.2 Effect of optimising initial data

The cases in Section 5.3.1 were repeated with the initial data optimised, so that the initial node positions were re-arranged to equidistribute the monitor (arc length) function. Figure 5.3 shows the effect on the initial grid of this equidistribution, for the 11-node case.

Figure 5.4 shows the solution grids for the optimised data, and it appears there is little difference from the unoptimised grids in Figure 5.1. In fact, looking at Figure 5.5, the orders of convergence for the L^2 and the boundary error are similar to those for the unop-

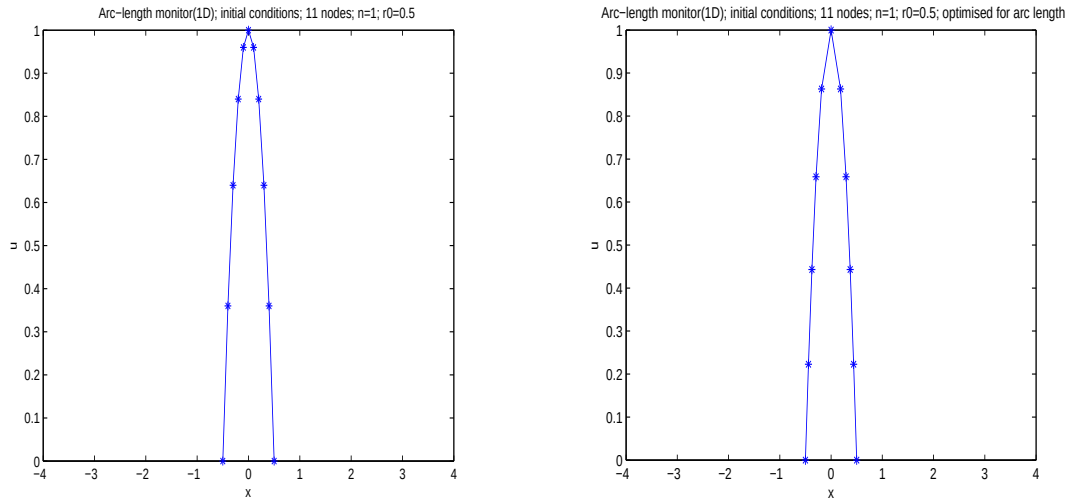

Figure 5.3: Initial grids for 11 nodes, n=1, unoptimised (left) and optimised (right) for arc-length monitor.

timised data in Figure 5.2. The plot of the distribution constants in Figure 5.5 is different from the unoptimised case in Figure 5.2, but still we can see the same general pattern of only staying constant at later times. Note that in Figure 5.5 (and some later ones), the distribution constants for the quarter-point and origin nodes are virtually identical, so the plots coincide.

Further cases with longer times and more refinement have shown similar convergence orders and the same pattern for the distribution constants, except that for 81 nodes in the optimised data case, there was a breakdown in the algorithm in determining the next grid positions, at $T = 0.259$. The grid just before the breakdown is shown in Figure 5.6, with a zoom on the circled nodes. The zoomed plot shows two nodes very close together, relative to the adjacent inter-node distances, and this may have caused an ill-conditioning in solving the matrix system in equations (5.2) and (5.3), where there would be a high relative ratio of successive values of ∇w_i, or it may be caused by the nodes tangling at the next step.

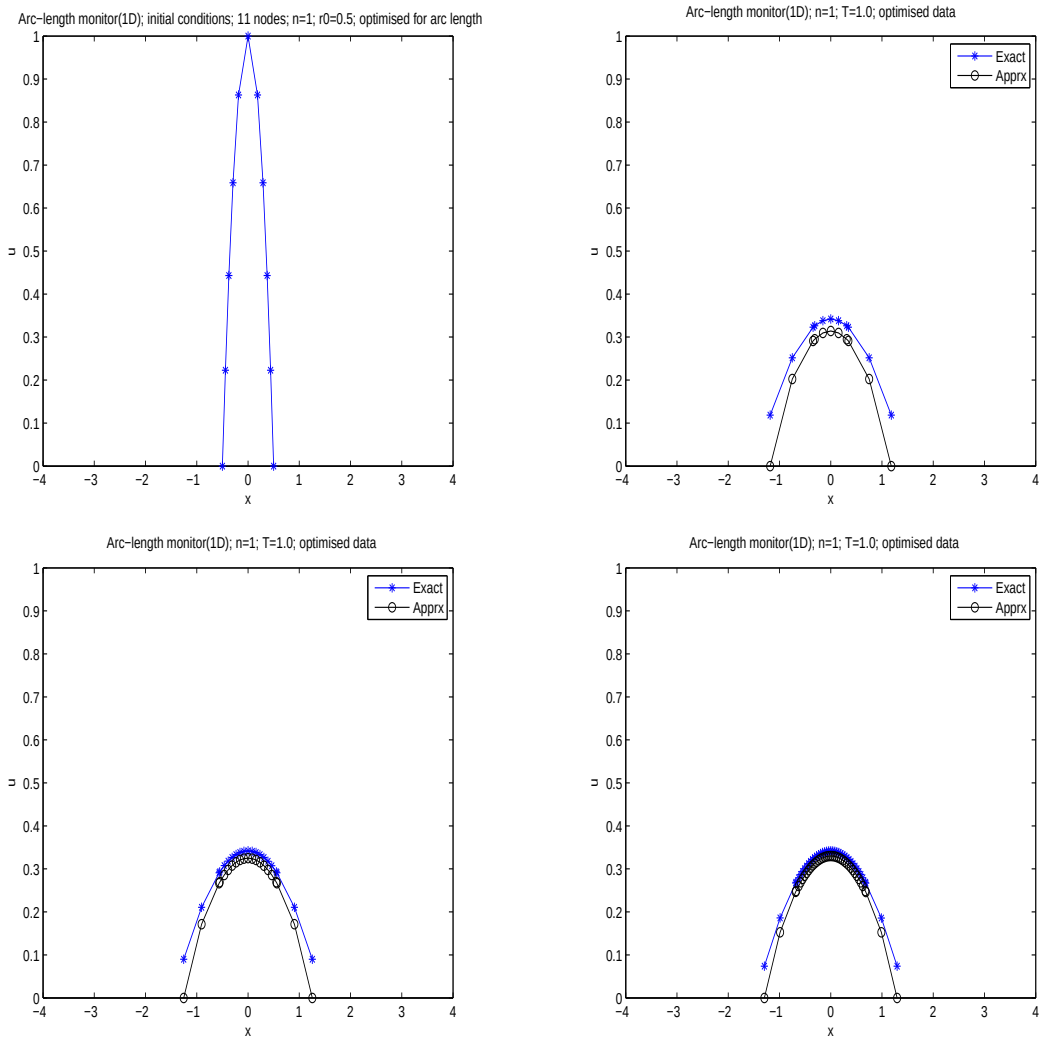

Figure 5.4: Grid evolution for ALE runs with optimised initial data, 11, 21 and 41 nodes. Graphs show initial grid, and exact (known) solution and approximation at T=1.0.

5.3.3 Methods of imposing boundary velocities

When using equations (5.2) and (5.3), the boundary velocities have been *influenced* by the normal boundary velocity estimate $\dot{\xi}$, but we have not actually forced them to be equal to $\dot{\xi}$. We now consider the effect of directly forcing this, by representing $\dot{x}_h$ as $\dot{\xi}_1 w_1 + (\dot{x}_2 w_2 + \cdots + \dot{x}_{N-1} w_{N-1}) + \dot{\xi}_N w_N$ (so forcing the boundary velocities), and then recasting the discrete form of equation (5.12), so it is only solved for internal nodes:-

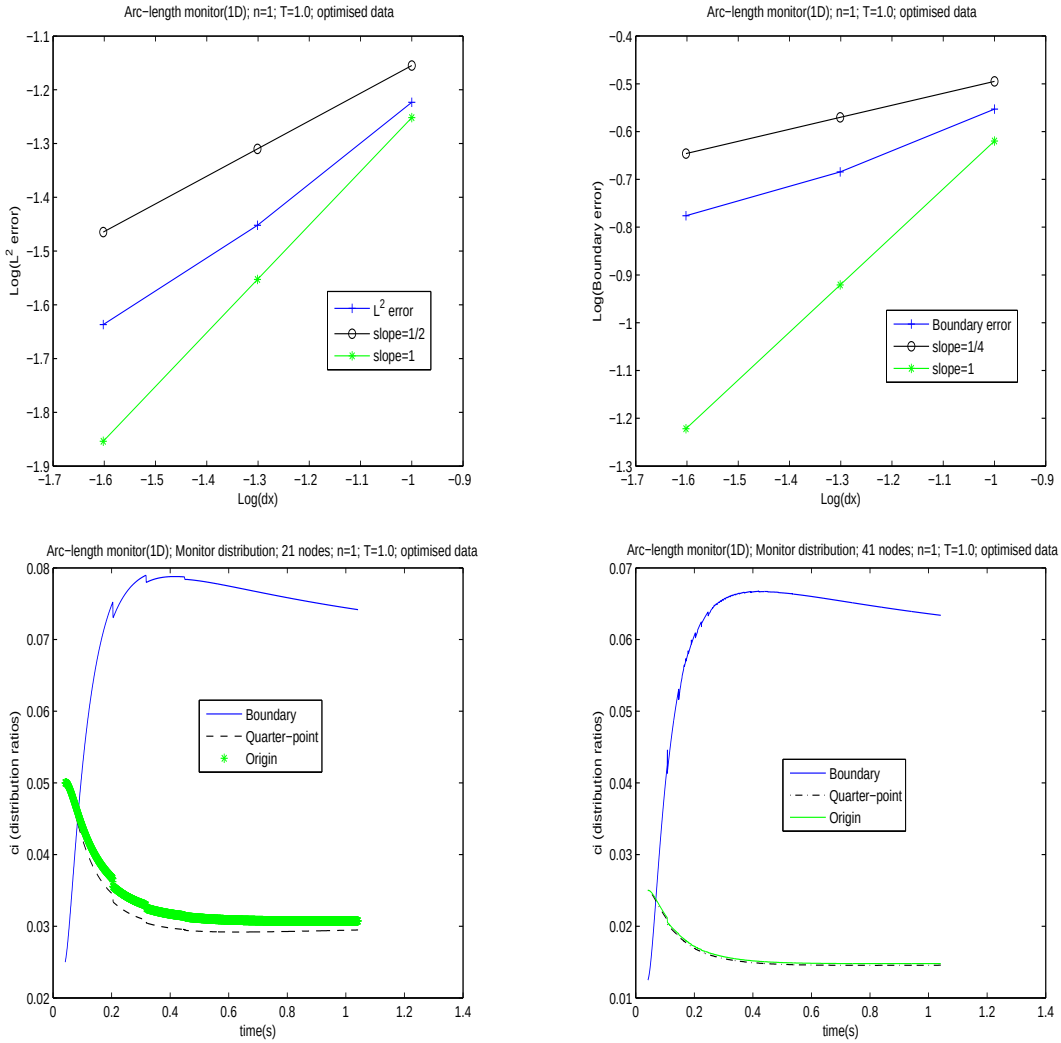

Figure 5.5: Convergence rates (solution and boundary error) and monitor distribution evolution for ALE runs with optimised initial data.

$$\int_{a(t)}^{b(t)} w_i \dot{x}_h dx = \int_{a(t)}^{b(t)} w_i \frac{\partial \phi_h}{\partial x} dx, \quad i = 2 \ldots N-1. \tag{5.20}$$

$$\Rightarrow \int_{a(t)}^{b(t)} w_i (\dot{\xi}_1 w_1 + \Sigma_{j=2}^{N-1} \dot{x}_j w_j + \dot{\xi}_N w_N) dx = \int_{a(t)}^{b(t)} w_i \frac{\partial \phi_h}{\partial x} dx, \quad i = 2 \ldots N-1. \tag{5.21}$$

$$\Rightarrow \int_{a(t)}^{b(t)} w_i (\Sigma_{j=2}^{N-1} \dot{x}_j w_j) dx = \int_{a(t)}^{b(t)} w_i \frac{\partial \phi_h}{\partial x} dx - \int_{a(t)}^{b(t)} w_i (\dot{\xi}_1 w_1 + \dot{\xi}_N w_N) dx, \; i = 2 \ldots N-1. \tag{5.22}$$

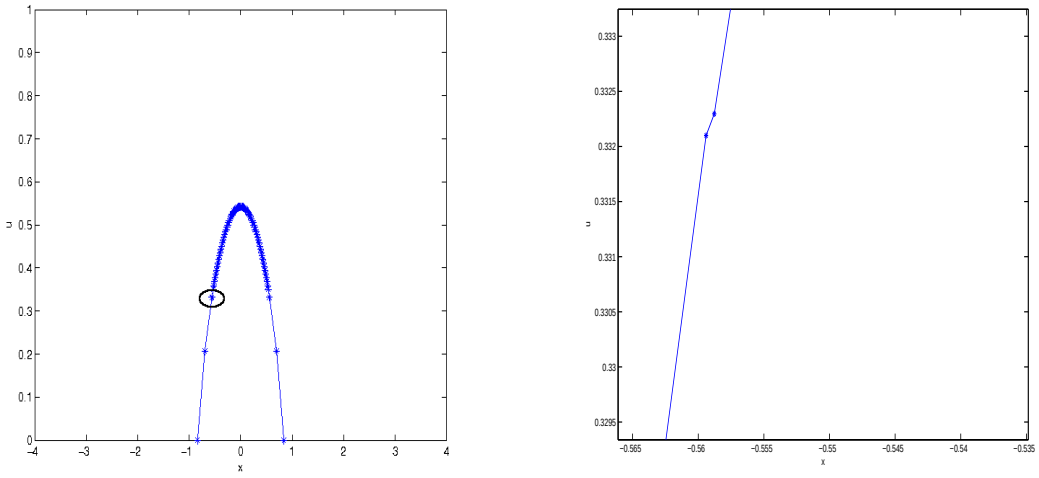

Figure 5.6: Grid and zoom-in at T=0.259 for ALE run with optimised initial data, 81 nodes.

5.3.3.1 Results

Mesh evolution, convergence and monitor distribution plots for unoptimised initial data are shown in Figures 5.7 and 5.8, and for optimised initial data in Figures 5.9 and 5.10. Looking at the unoptimised plots, we can see the accuracy has improved when compared with the results in Section 5.3.1, and the convergence plots now show orders of approximately 1.5 for the L^2 error and 1.25 for the boundary error. When the initial data is optimised, Figures 5.9 and 5.10 show a further improvement, with convergence orders of 2 and 1.5 for the solution error (L^2) and the boundary error respectively. The order for the solution error thus agrees with the original study [4], though the mesh error (for BHJx) is slightly less. As for the distribution constants c_i, we see a different pattern compared to runs where the boundary velocity is not forced (in Section 5.3.1), though they are still not remaining constant for early times. The algorithm has proved more robust - longer times and further mesh refinement have produced results consistent with the above. Given the improvements these changes have made, we assume in the rest of Chapter 5 that unless stated otherwise, $\dot{x}$ is always applied directly.

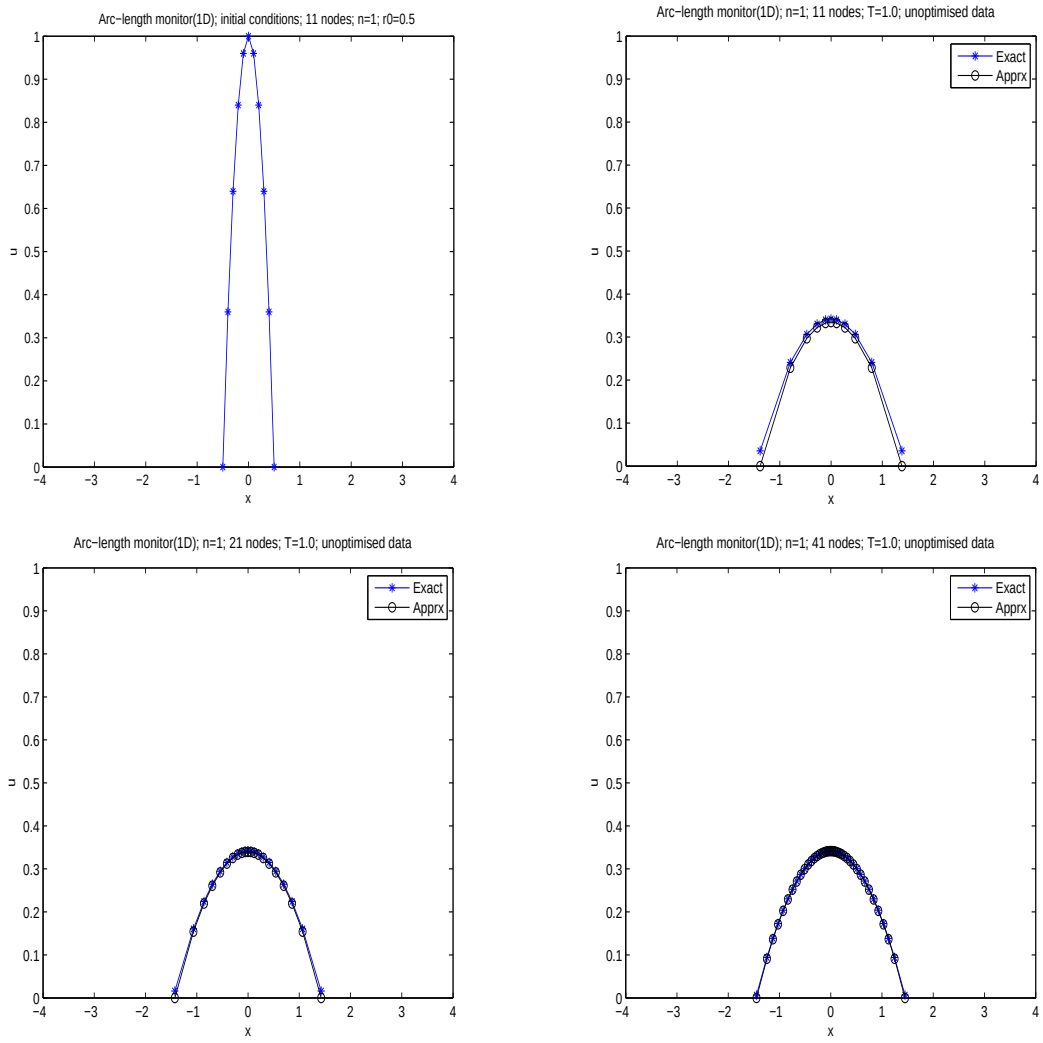

Figure 5.7: Grid evolution for ALE runs with unoptimised initial data, forced boundary velocities, 11, 21 and 41 nodes. Graphs show initial grid (top left), and exact (known) solution and approximation at T=1.0.

5.3.4 Strong vs Weak boundary conditions

We discuss here the effect of only applying the boundary conditions weakly, so equation (2.4) (in 1D form for the PME) is now solved for all nodes, but $\int_{a(t)}^{b(t)} w_i \frac{\partial}{\partial x}(u^n \frac{\partial u}{\partial x})\, dx$ is evaluated as $-\int_{a(t)}^{b(t)} \frac{\partial w_i}{\partial x}(u^n \frac{\partial u}{\partial x})\, dx$, so effectively weakly imposing $\left[w_i u^n \frac{\partial u}{\partial x}\right]_{a(t)}^{b(t)}$ as zero.

Mesh plots for 11-node runs to T=0.1 and 1.0, for optimised initial conditions, are shown

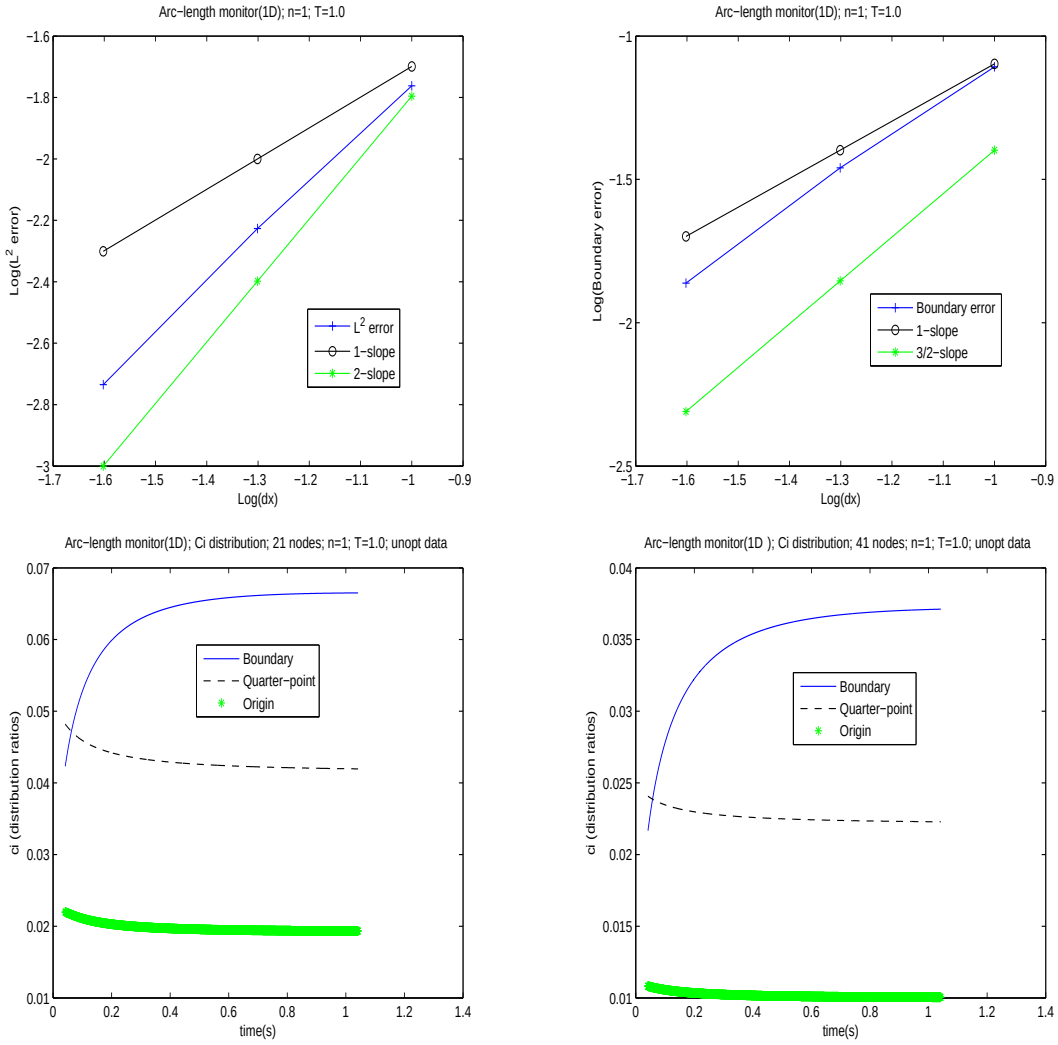

Figure 5.8: Convergence rates (solution and boundary error) and monitor distribution evolution for ALE runs with unoptimised initial data and forced boundary velocities.

in Figure 5.11. We can see the grid has lifted up and eventually degenerated, so this change has not produced a desirable effect, and grid refinement had no effect on this result. Consequently, we only consider strong boundary conditions for u in subsequent tests.

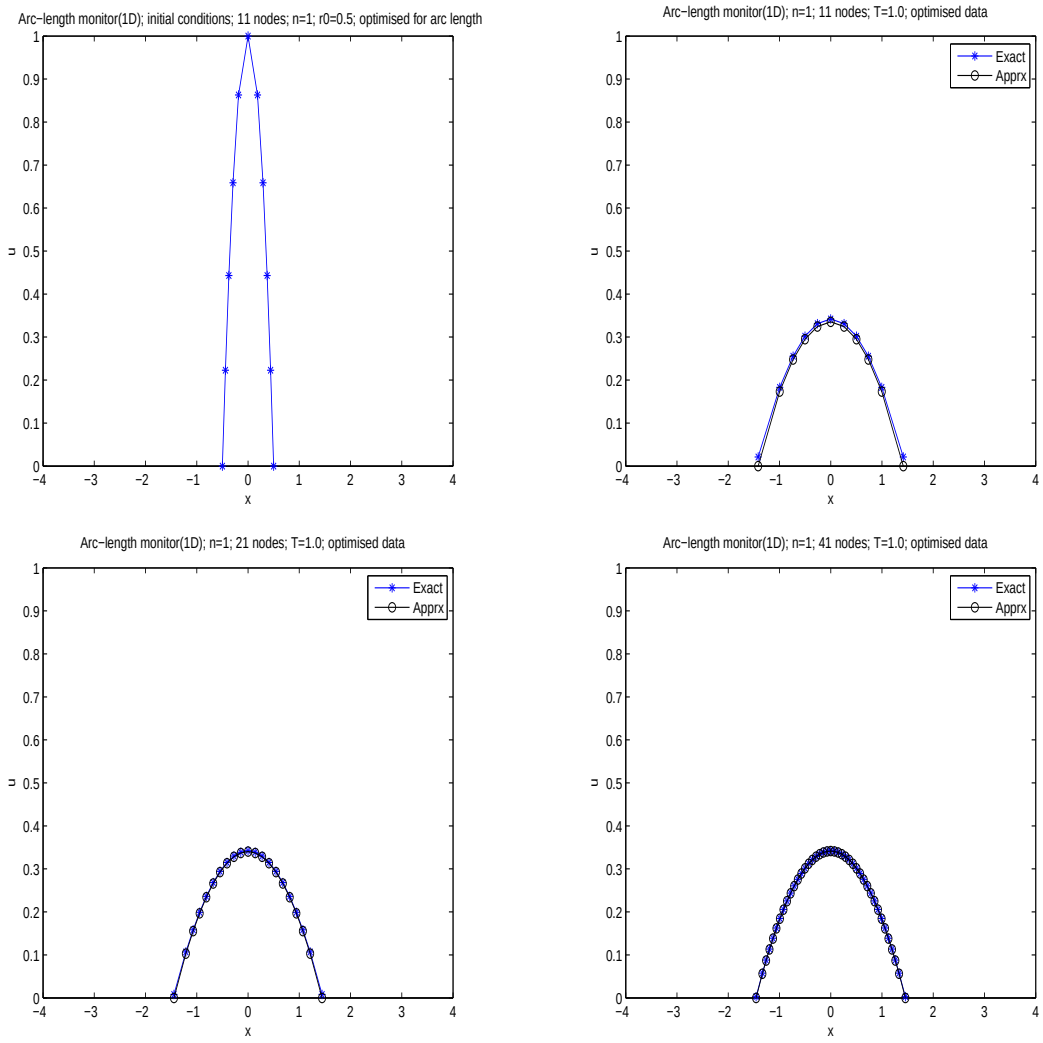

Figure 5.9: Grid evolution for ALE runs with optimised initial data, forced boundary velocities, 11, 21 and 41 nodes. Graphs show initial grid, exact (known) solution and approximation at T=1.0.

5.3.5 Accuracy of third derivative terms

If we expand the second derivative terms of the porous medium equation PDE in equation (5.1):-

$$\frac{\partial(u^n \frac{\partial u}{\partial x})}{\partial x} = u^n \frac{\partial^2 u}{\partial x^2} + n u^{n-1} \frac{\partial u}{\partial x}, \tag{5.23}$$

then another method of evaluating q in equation (5.17) is to use the nodal values of u to create a set of cubic splines [68] that estimate u in the PDE domain. The first and

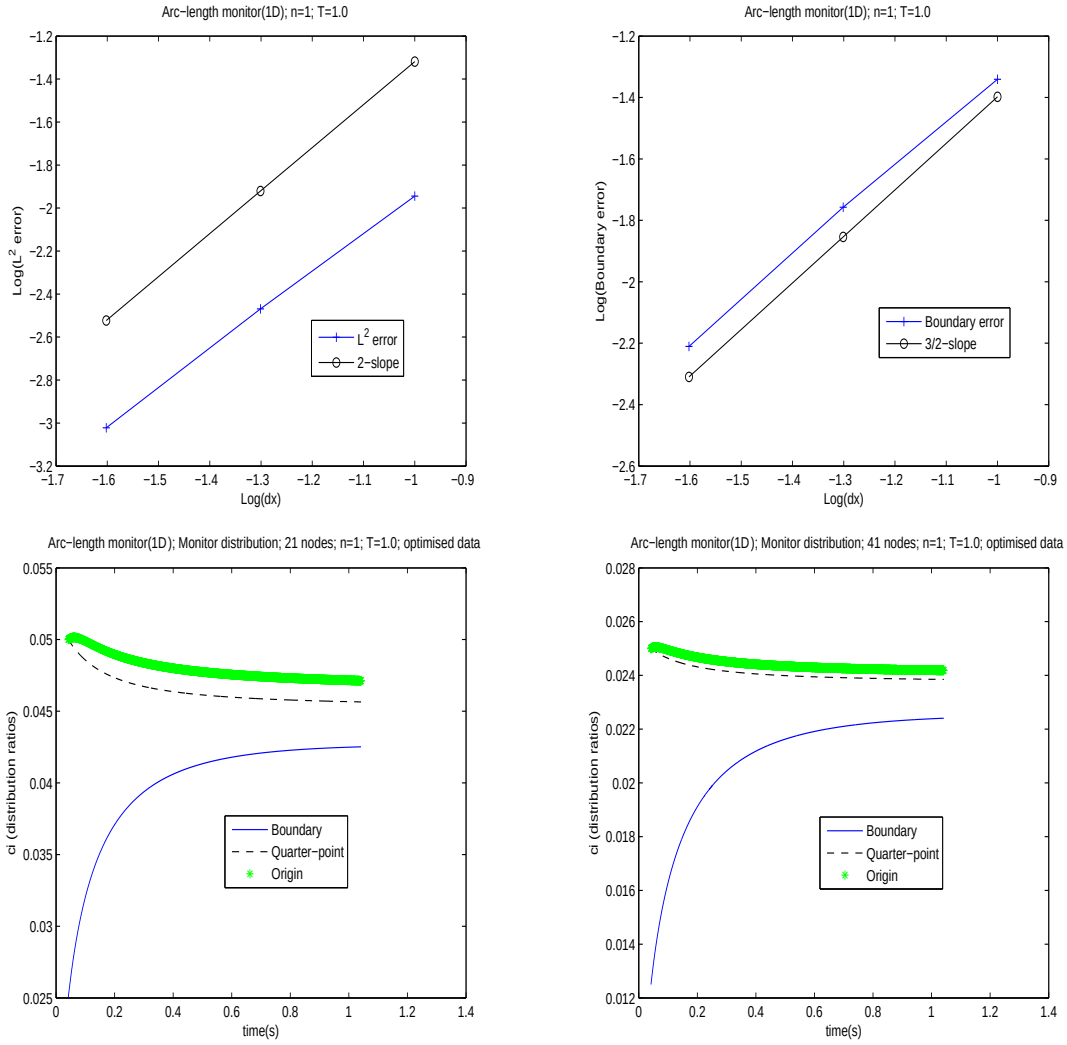

Figure 5.10: Convergence rates (solution and boundary error) and monitor distribution evolution for ALE runs with optimised initial data and forced boundary velocities.

second derivatives of these splines, and the nodal values themselves, can then be used to estimate the nodal values of Lu from equation (5.23). This method was tried, as a possible alternative to the approach used to find the third derivative terms, as evaluated by the weighted method described in Section 5.2. However, it proved not to be a robust method, giving non-symmetric results and causing mesh tangling. An example plot is shown in Figure 5.12 for 41 nodes to a run time of T=0.0457, using optimised data - the mesh completely collapsed shortly after this time.

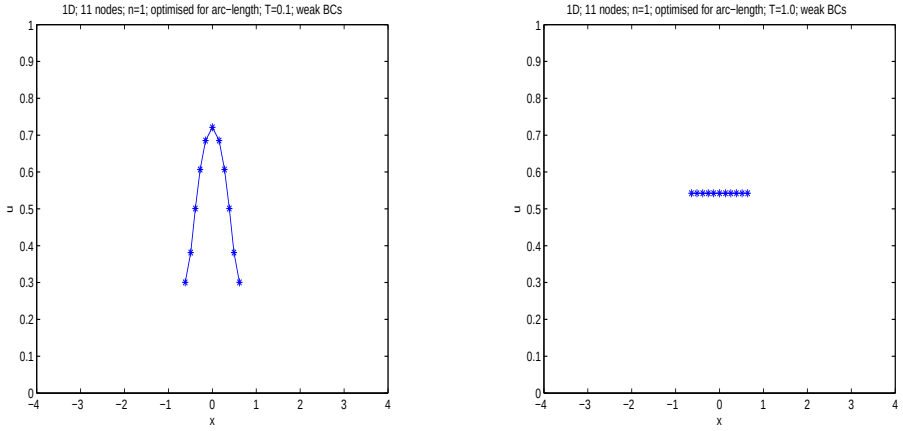

Figure 5.11: Grids at T=0.1 and 1.0 for ALE run, with optimised initial data and weak boundary conditions, 11 nodes.

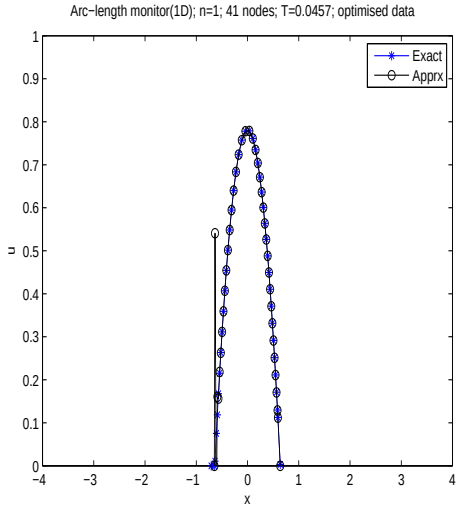

Figure 5.12: Grid and exact (known) solution at T=0.0457 for ALE run with optimised initial data, 41 nodes, where cubic splines were used to calculate third derivative terms.

5.4 ALE+ (Forcing the distribution constants)

5.4.1 Mathematical Description

In the first part of this chapter on the arc-length monitor, we have seen that the algorithm has been successful in terms of accuracy and robustness, under certain conditions, but the c_i have not actually remained constant throughout the run. We now consider adding a

constraint to the method that forces this. The value of u is updated in the usual way via the ALE equation (2.4) and the value of x is also updated from $\dot{x}$. However, that value of u is now taken as the initial value in an additional algorithm, where x is fixed, but we let the nodal values of u vary, until the c_i regain their starting value. Specifically, we solve these equations:-

$$\int_{\Omega(t)} w_i m(v) dx = c_i \theta(t),\ i = 1,2 \ldots N, \tag{5.24}$$

where $\Omega(t), w_i, c_i$ and $\theta(t)$(using the *updated* value, via $\dot{\theta}$ from equation (5.3)) are all constant, but u (and so $v(=\frac{\partial u}{\partial x})$) varies. If we write $F_i(u) = (\int_{\Omega(t)} w_i m(v) dx) - c_i \theta(t),\ i = 1,2 \ldots N$, then our goal is to solve the system of equations:-

$$F_i(u) = 0,\ i = 1,2 \ldots N. \tag{5.25}$$

Writing $\mathbf{F}(u) = (F_1(u), F_2(u), \ldots F_N(u))^T$ and $\mathbf{u} = (u_1, u_2, \ldots u_N)^T$, solving (5.25) is equivalent to solving the single vector equation:-

$$\mathbf{F}(\mathbf{u}) = \mathbf{0}. \tag{5.26}$$

A Newton-Krylov method [13] was used to solve this system, which only requires us to supply a function $\mathbf{F}$ of a vector function $\mathbf{u}$ and an initial guess $\mathbf{u}_0$. The derivative-product $\mathbf{F}'(\mathbf{u})\mathbf{h}$ is then approximated by:-

$$\mathbf{F}'(\mathbf{u})\mathbf{h} \approx \frac{\mathbf{F}(\mathbf{u} + \sigma\mathbf{h}) - \mathbf{F}(\mathbf{u})}{\sigma}, \tag{5.27}$$

for a scalar σ [13]. This algorithm was implemented with the SUNDIAL suite [42], which performs the approximation in (5.27).

5.4.2 Basic Cases

The cases in Section 5.3.1 were repeated for the ALE+ method. The grid plots at $T = 1.0$ are shown for 11, 21 and 41 nodes in Figure 5.13, and the solution convergence and monitor evolution plots are shown in Figure 5.14. Comparing these with the ALE unoptimised plots in Figures 5.7 and 5.8, we can see that the approximation is not converging to the solution for ALE+ as fast as ALE, when refining the mesh from 11 nodes to 21 nodes. However, on further refinement, the order of the ALE+ method does increase, and nearly to order 3 in the case of the L^2 solution error. As for the distribution constants, we can see they are staying constant for the 11-node run, but on further refinement, they are showing the same pattern as in Figure 5.8. We can see the order of convergence has improved as the distribution evolution has got poorer (the distribution constants not staying constant), and the mesh has been refined. At the higher node runs though (21 onwards), we can see that, comparing the ALE+ runs with the ALE runs, just attempting to maintain the distribution by using equation (5.26) has improved the order of convergence.

More pertinent however, is that the distribution constants are not staying constant, even though this has supposedly been forced by solving equation (5.26). In fact, the default tolerance in the SUNDIAL suite [42] is approximately 1×10^{-6}, so that if $|\mathbf{F}|$ is less than this tolerance in equation (5.26), it is considered to be zero, and the Newton-Krylov iteration terminates. Experiments were done to lower this tolerance, to see if the distribution evolution could be improved. The result of one such experiment is shown in Figure 5.15 for 21 nodes, where the tolerance was lowered to 1×10^{-9}. We can see the approximation is worse than that in Figure 5.13 for 21 nodes, but the distribution constants are now staying constant. The 41 and 81 node cases would not run at this tolerance though - causing a failure in the SUNDIAL suite to reach convergence.

Concluding here, we can see that just attempting to force a distribution to be maintained has gained an order of convergence for the solution error, but if we actually do force the c_i to be constant, the algorithm fails.

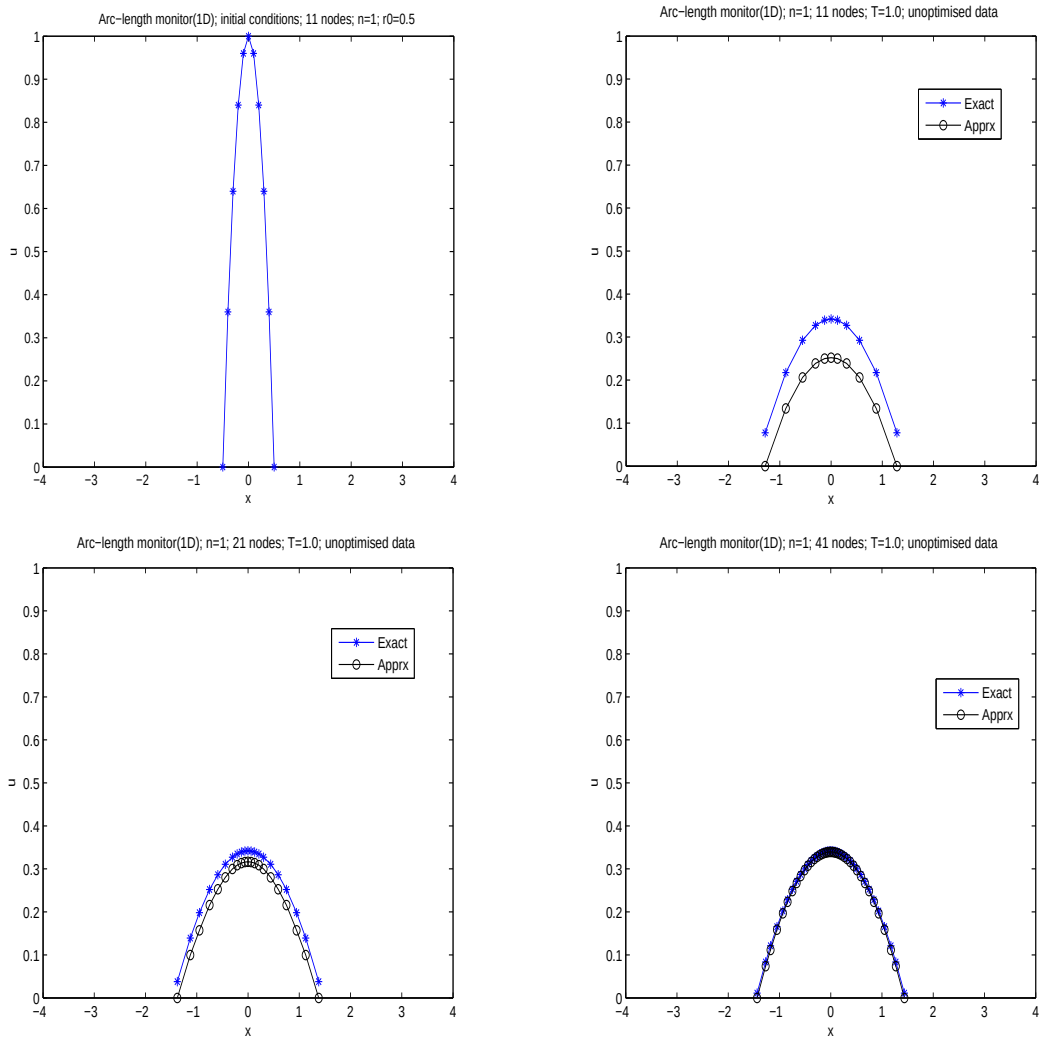

Figure 5.13: Grid evolution for ALE+ runs with unoptimised initial data, 11, 21 and 41 nodes. Graphs show initial grid, and exact (known) solution and approximation at T=1.0.

5.4.3 Effect of optimising initial data

The cases in Section 5.3.1 were repeated for the ALE+ method, with the initial data optimised, so that the initial node positions were re-arranged to equidistribute the monitor (arc length) function. When these cases were run with the default SUNDIAL tolerance of 1×10^{-6}, as described in Section 5.4.2, the distribution constants had a poor evolution, showing a similar pattern to those in Figure 5.10. They were therefore (successfully) repeated at a lower tolerance of 1×10^{-9}, following the results in Section 5.4.2. For these

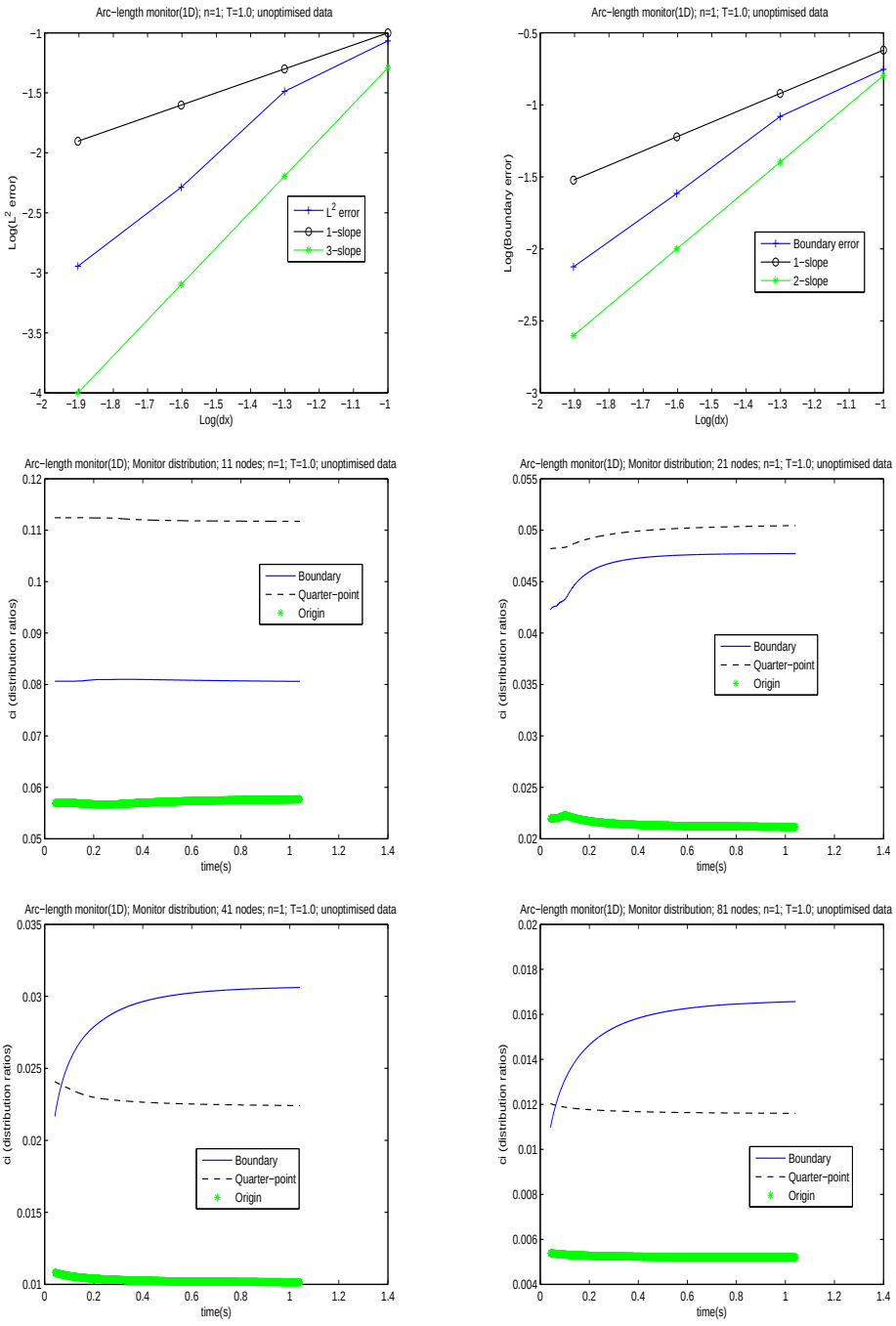

Figure 5.14: Convergence rates (solution and boundary error) and monitor distribution evolution for ALE+ runs with unoptimised initial data.

reduced tolerance runs, the grids at $T = 1.0$ are shown for 11, 21 and 41 nodes in Figure 5.16, and the solution convergence and monitor evolution plots are shown in Figure 5.17. Comparing these with the unoptimised case, we can see the mesh approximation is

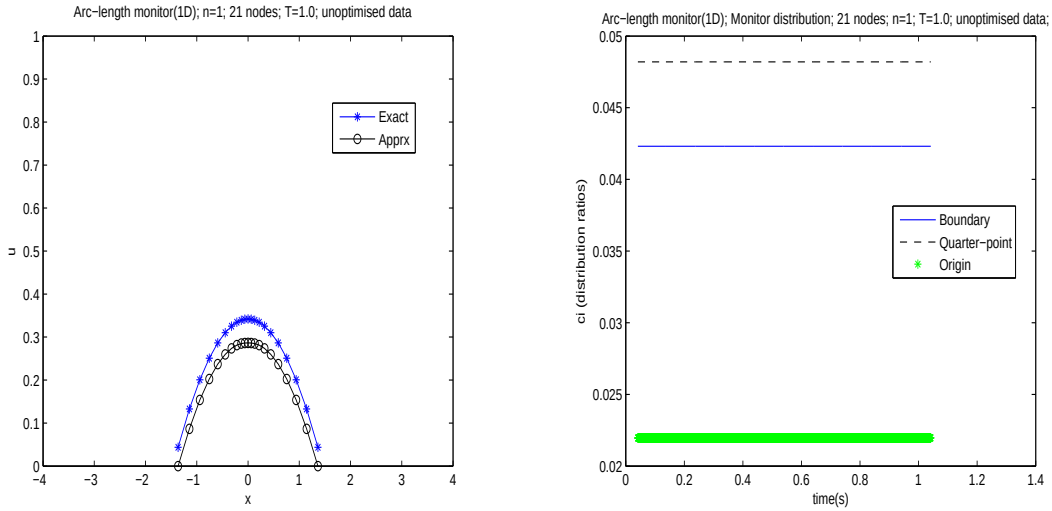

Figure 5.15: Grid and monitor distribution evolution for ALE+ run with unoptimised initial data, 21 nodes. SUNDIAL tolerance lowered to 1×10^{-9}.

poorer, now being only of order 1, but the distribution constants are now staying constant. Following these successful runs to maintain the distribution, further cases were run to test the robustness of the method. Using the same SUNDIAL tolerance of 1×10^{-9}, the 161 nodes was run. This ran up to $T = 0.4375$, at which point the SUNDIAL suite failed to reach convergence. The SUNDIAL tolerance was then increased until the 161 node case ran to $T = 1.0$. At a tolerance of 3×10^{-9}, the run did not reach $T = 1.0$, but at 4×10^{-9}, it did. The distribution evolution for these two cases (1×10^{-9} and 4×10^{-9}) is shown in Figure 5.18. We can see a slight increase in c_1 (boundary constant) for the lower tolerance, but a higher increase for the higher tolerance, even though that ran for the longer time. In both cases though, the relative increase is lower than we see for 41 and 81 nodes in the case where the initial data is unoptimised - see Figure 5.14.

Finally, the 41-node case was run for a longer time of $T = 10.0$ at the lower tolerance of 1×10^{-9}, and the grid and monitor distribution evolution are shown in Figure 5.19.

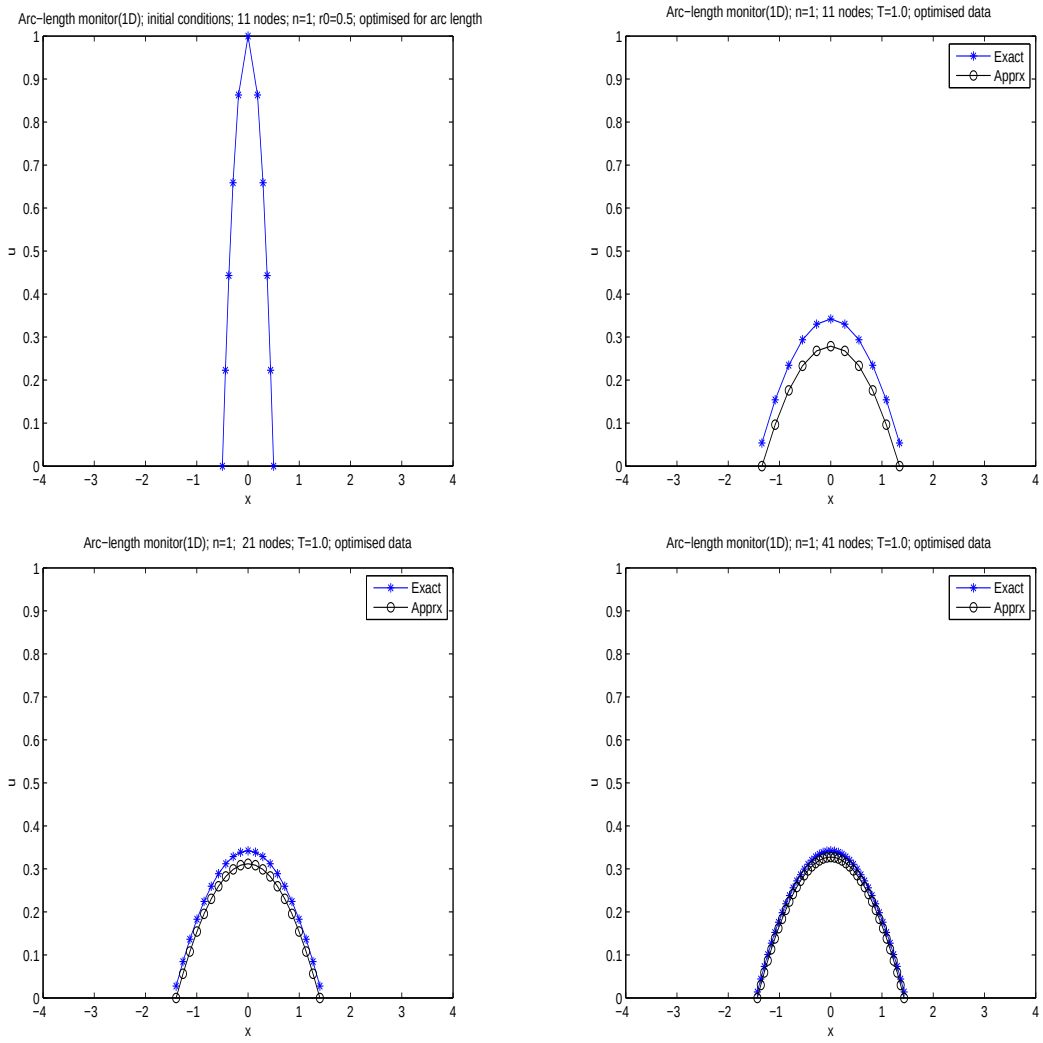

Figure 5.16: Grid evolution for ALE+ runs with optimised initial data, 11, 21 and 41 nodes. Graphs show initial grid, and exact (known) solution and approximation at T=1.0. SUNDIAL tolerance lowered to 1×10^{-9}.

Comparing this with the $T = 1.0$ run for the same 41-node case, shown in Figures 5.16 and 5.17, we can see the monitor distribution has again been maintained.

Concluding here, we can see it is possible to maintain an equidistribution, though it does involve an extra parameter - the SUNDIAL tolerance. We also note the effect of forcing an equidistribution is to lose one order of convergence. On the other hand, we can say allowing the distribution to deviate from an equidistribution has gained an order of convergence.

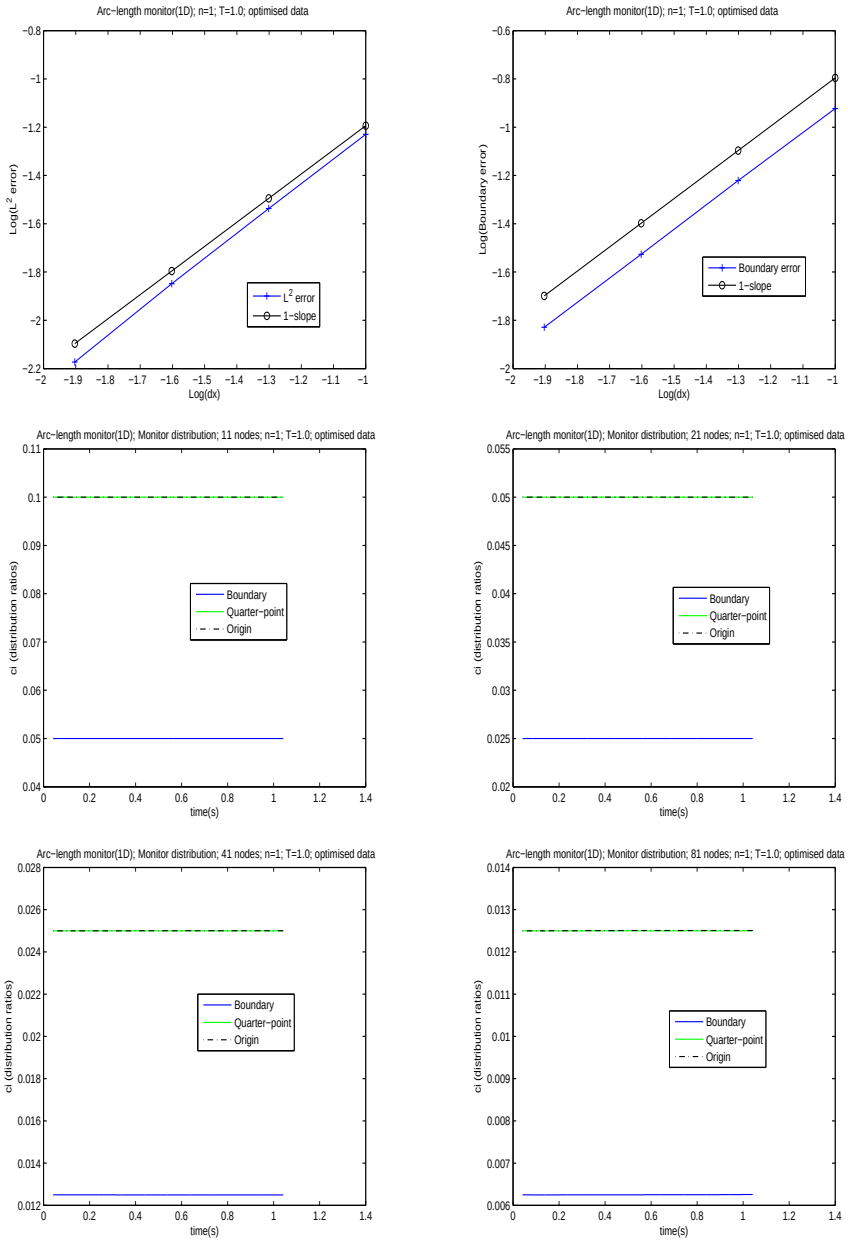

Figure 5.17: Convergence rates (solution and boundary error) and monitor distribution evolution for ALE+ runs with optimised initial data. SUNDIAL tolerance lowered to 1×10^{-9}.

5.5 n=3 cases

We consider here the cases where the porous medium equation [92] parameter (n) is set to 3. In this case, the similarity solution has an infinite slope at the moving boundary, which

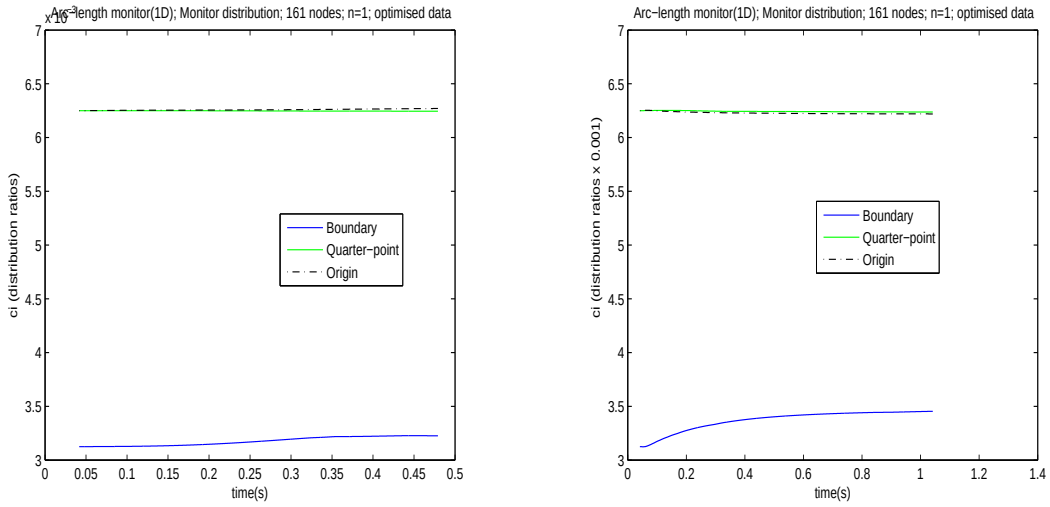

Figure 5.18: Monitor distribution evolution for ALE+ runs with optimised initial data, 161 nodes, SUNDIAL tolerance of 1×10^{-9} (partial run) and SUNDIAL tolerance of 4×10^{-9} (full run to T=1.0).

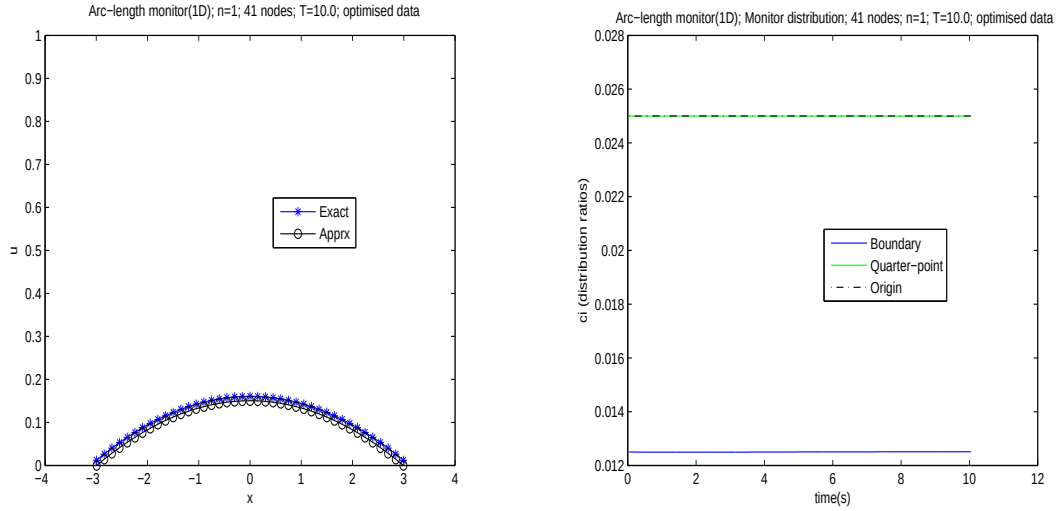

Figure 5.19: Grid and monitor distribution evolution for ALE+ runs with optimised initial data, 41 nodes, T=10.0. SUNDIAL tolerance of 1×10^{-9}.

makes numerical simulation more of a challenge than for the $n = 1$ case. Furthermore, this similarity solution is a stable attractor, so that the solution tends towards it, for all initial data.

As with the $n = 1$ cases, we will look at the ALE and ALE+ codes, running against

unoptimised and optimised initial conditions. The ALE code has the boundary velocities imposed directly, as described in Section 5.3.3. The initial conditions for 21 nodes are shown in Figure 5.20, where we can see much steeper slopes at the boundary than for $n = 1$.

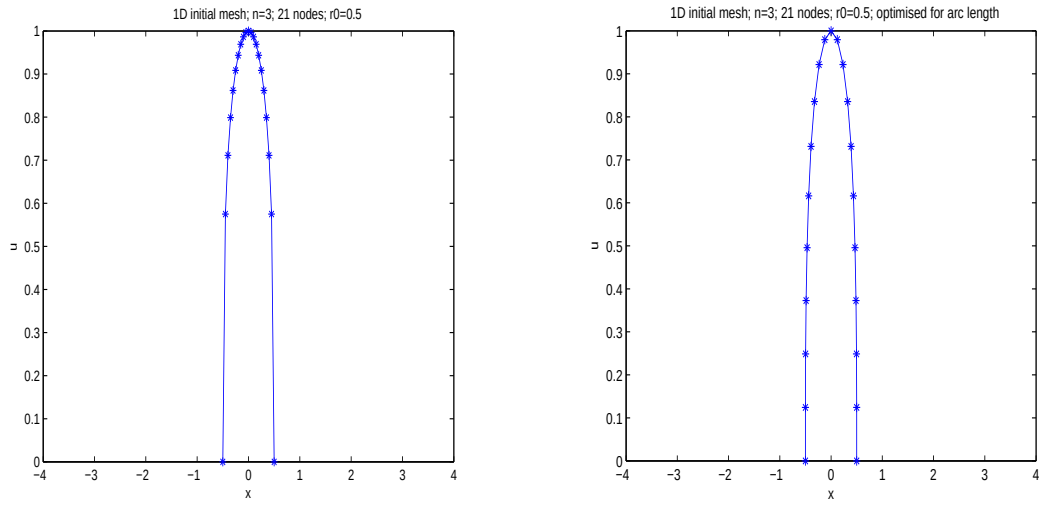

Figure 5.20: Initial conditions for $n = 3$, 21 nodes, unoptimised and optimised initial conditions.

5.5.1 ALE with unoptimised initial conditions

The cases in Section 5.3.1 were repeated for $n = 3$, but with 11, 21, 41 and 81 nodes. Although the L^2 approximation error decreased as the grid was refined from 11 nodes to 41 nodes, it actually increased from 41 nodes to 81 nodes. The grids for 41 and 81 nodes are shown in Figure 5.21, along with the monitor distribution evolution for these two cases. We can see the internal nodes "bunching" towards the centre, so that the approximation is actually getting worse as this bunching intensifies. It is also clear this behaviour is preventing the monitor distribution being maintained. We can see some similarity here with the ALE($n = 1$) case shown in Figure 5.6, where the grouping of nodes may have caused an ill-conditioning problem.

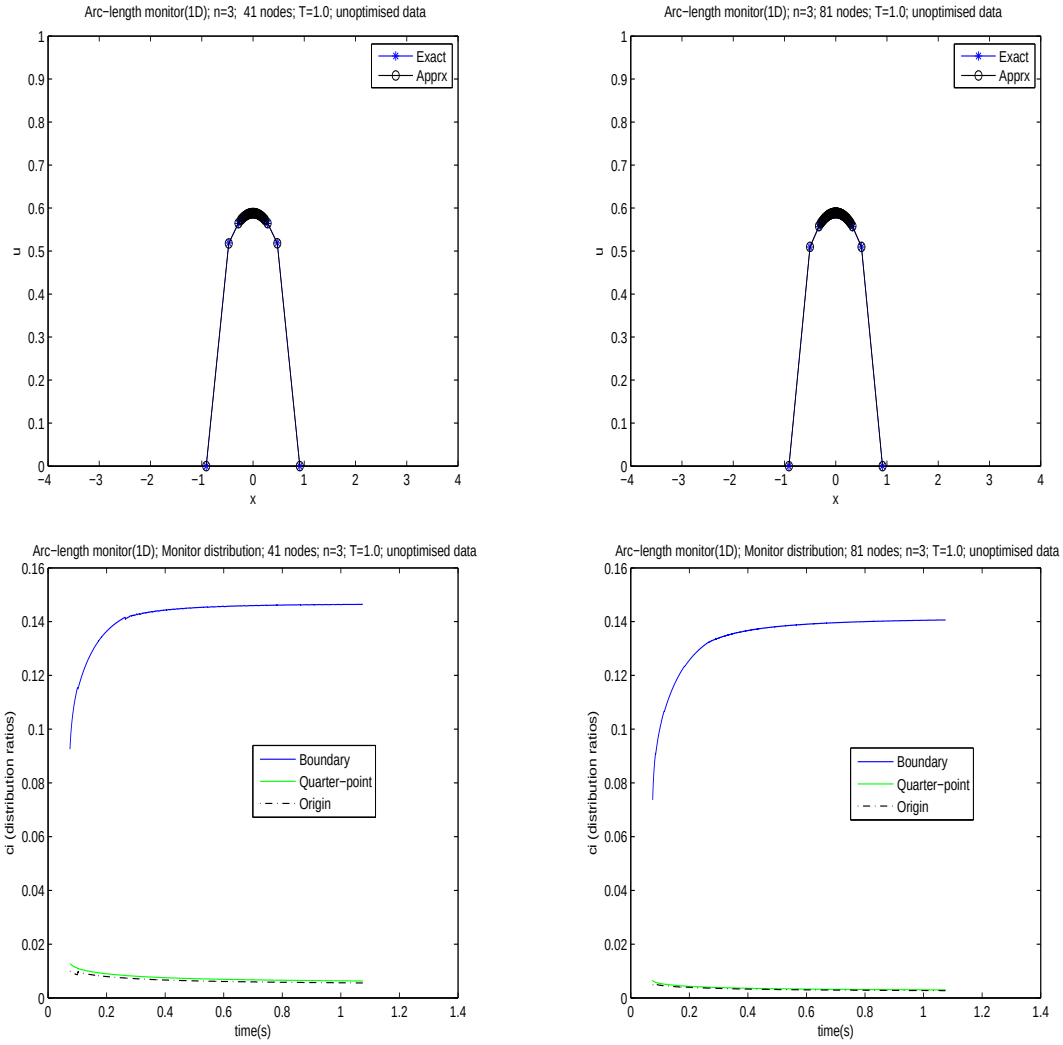

Figure 5.21: Meshes at T=1.0 and monitor distribution evolution for ALE runs with unoptimised initial data, 41 and 81 nodes, n=3.

5.5.2 ALE with optimised initial conditions

The cases in Section 5.5.1 were repeated with the initial conditions optimised, to equidistribute arc-length. Only the 11 and 21 node cases ran all the way through to $T = 1.0$. In the case of 41 and 81 nodes, there was a breakdown in the algorithm in determining the next grid positions (not in the SUNDIAL suite). The final grid for 21 nodes, and the grid for 41 nodes, just before the algorithm broke down, are shown in Figure 5.22. We can see the "bunching" behaviour again, as we saw for the unoptimised runs, but we also see a

steeper mesh here, which has caused a loss of robustness in the method. It is clear from the mesh plots that the monitor distribution (c_i) is not being maintained, as the initial data had an equidistributed arc length.

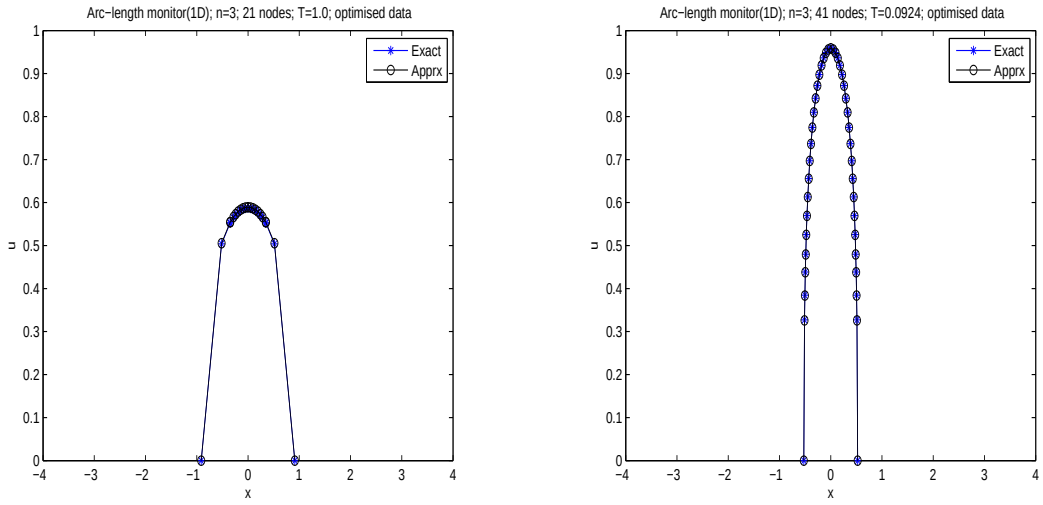

Figure 5.22: Meshes at T=1.0 for 21 nodes and T=0.0924 for 41 nodes, for ALE runs with optimised initial data, n=3.

5.5.3 ALE+ with unoptimised initial conditions

The cases in Section 5.5.1 were repeated for the ALE+ code, so we now try to force the c_i to be constant. In this case, the 11, 21 and 41 node cases ran all the way through to $T = 1.0$, but the 81-node mesh tangled (element lengths became negative), and there was then a failure in the SUNDIAL suite to converge. The convergence rates, and meshes for 21 and 41 nodes are shown in Figure 5.23. As with the $n = 1$ cases ran for the ALE+ code in Section 5.3.1, just attempting to maintain the distribution has improved the order of convergence (though only up to the 41 node case), but also as with the $n = 1$ case, we have not actually maintained the distribution, as can be seen in Figure 5.24. These runs were all done with the default SUNDIAL tolerance of 1×10^{-6} - attempts to lower this tolerance caused the SUNDIAL suite to fail before $T = 0.16$ was reached, for all cases.

As with the ALE code for the $n = 3$ cases, we can see a "bunching" of nodes towards the centre in the mesh plots (Figure 5.23), though there are fewer nodes doing this, than there were in the ALE case. This is reflected in the monitor evolution plot for 21 nodes in Figure 5.24, where we can see the quarter-point value has increased. This difference can also be seen in Figure 5.25, showing the 81-node case at $T = 0.2$, shortly before the SUNDIAL suite failed to converge.

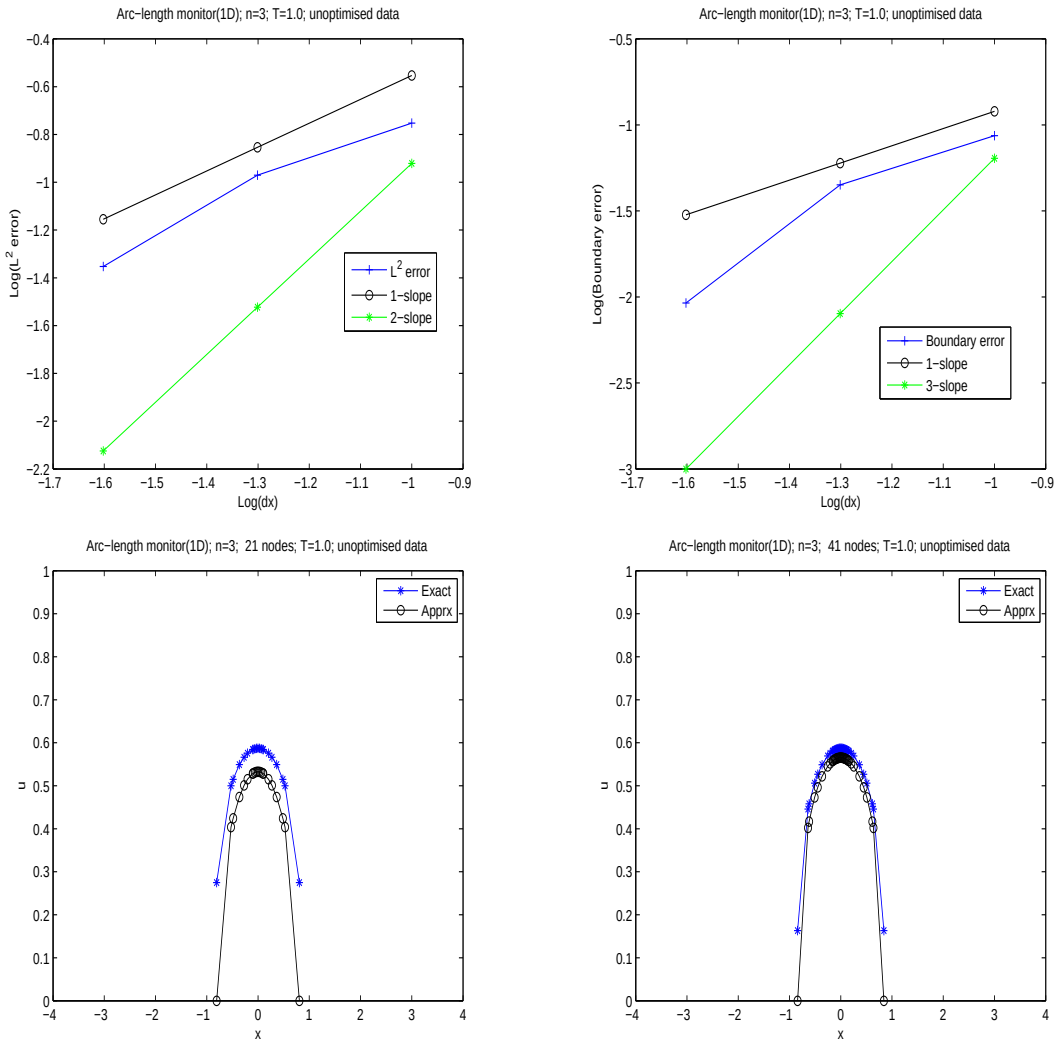

Figure 5.23: Convergence rates (solution and boundary error) and meshes at T=1.0 for 21 nodes and 41 nodes for ALE+ runs with unoptimised initial data, n=3.

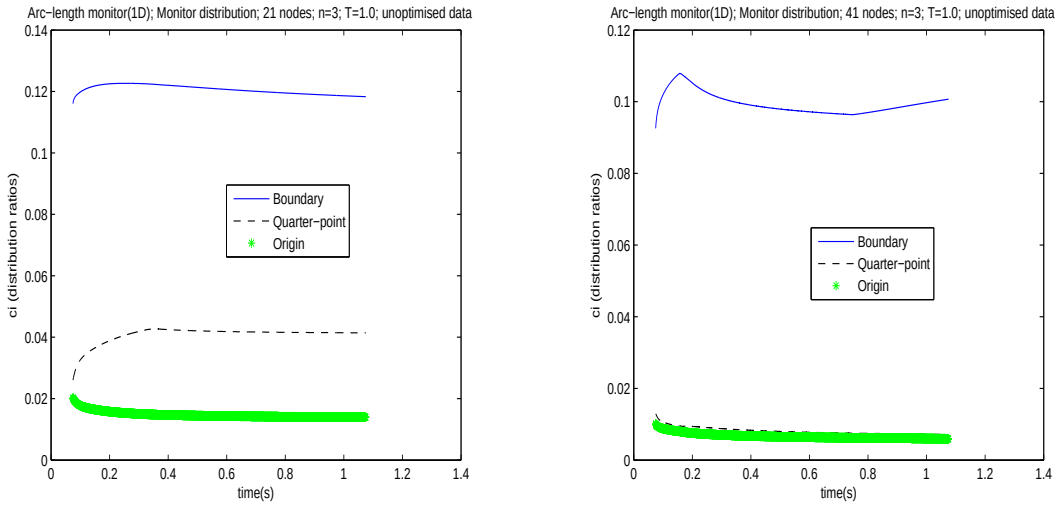

Figure 5.24: Monitor distribution evolution for ALE+ runs with unoptimised initial data, 21 and 41 nodes, n=3.

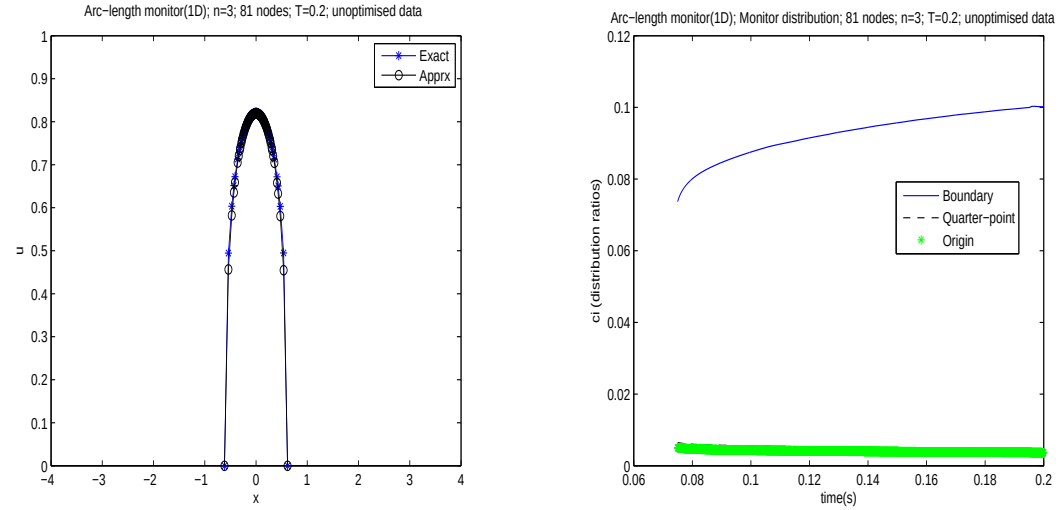

Figure 5.25: Mesh and monitor distribution evolution for ALE+ run with unoptimised initial data, 81 nodes at T=0.2, n=3.

5.5.4 ALE+ with optimised initial conditions

The cases in Section 5.5.2 were repeated for the ALE+ code, so we now try to force the c_i to be constant, with the initial conditions optimised, so we are starting with the arc-length being equidistributed. For these cases, the SUNDIAL suite failed to converge

after a very short time, so only runs up to T=0.01 for 11, 21 and 41 nodes are discussed here. The meshes at $T = 0.01$ and two of the monitor distributions are shown in Figures 5.26 and 5.27. The monitor distribution has been maintained. The meshes show a poor approximation though, and in fact, the L^2 solution error actually increased as the mesh was refined.

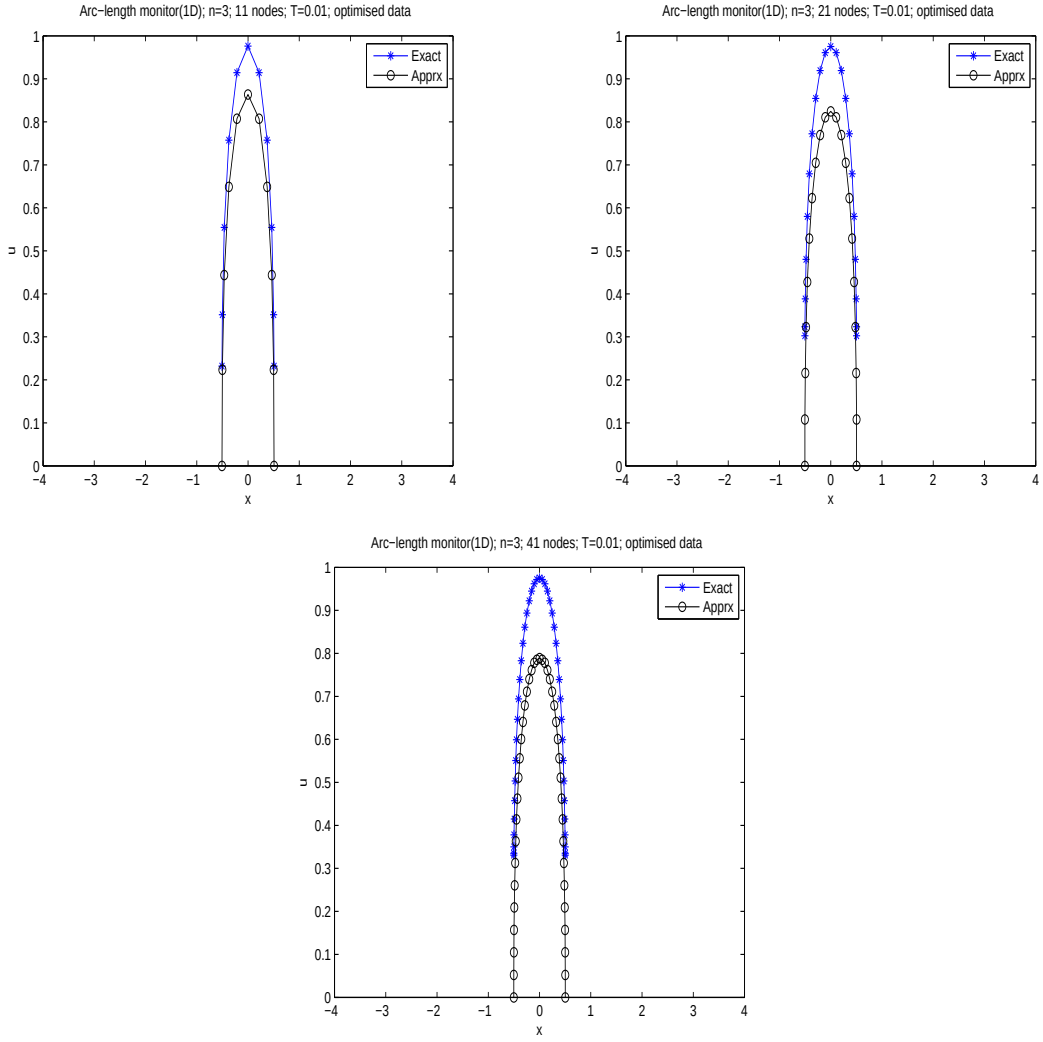

Figure 5.26: Meshes at T=0.01 for 11, 21 and 41 nodes for ALE+ runs with optimised initial data, n=3.

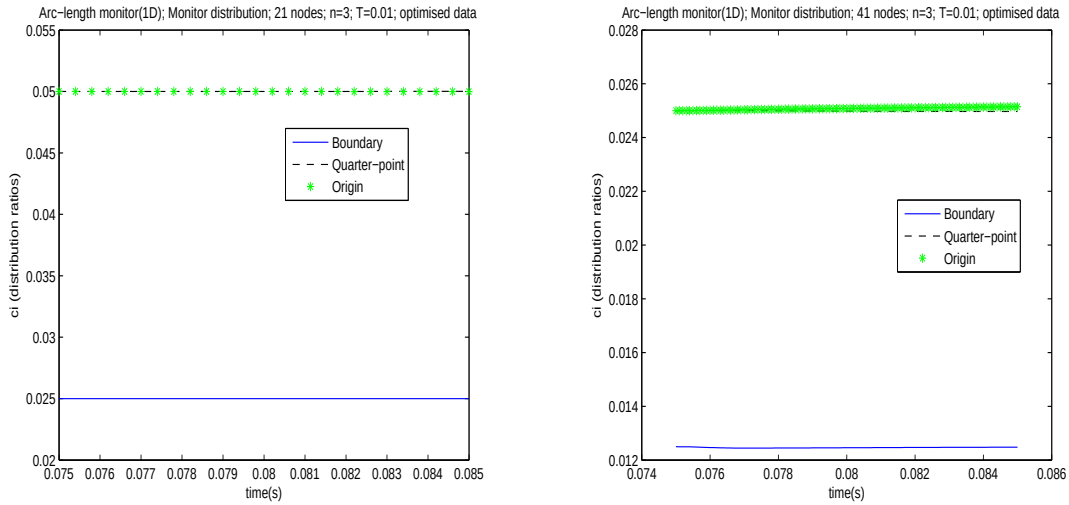

Figure 5.27: Monitor distribution evolution for ALE+ runs with optimised initial data, 21 and 41 nodes, n=3.

5.6 2D Cases

In 2D, preliminary results show a mesh pattern similar to that in the 1D case, and a consequent lack of robustness. A plot of the mesh is shown in Figure 5.28, for the $n = 1$ case, 545 nodes, running to T=0.1, where we can see the third annulus of cells in from the boundary has become compressed. A zoomed plot of the mesh for 8321 nodes, also running to T=0.1 and for $n = 1$, is shown in Figure 5.29, which shows the annulus compressing even further. This compression causes mesh quality problems, leading to poor accuracy, and can be compared with Figures 5.9 and 5.22 for the 1D case. For the $n = 3$ case, this compression has led to mesh tangling and subsequent collapse in this annulus, this tangling starting at the four “45-degree” lines, where there is a change of mesh geometry.

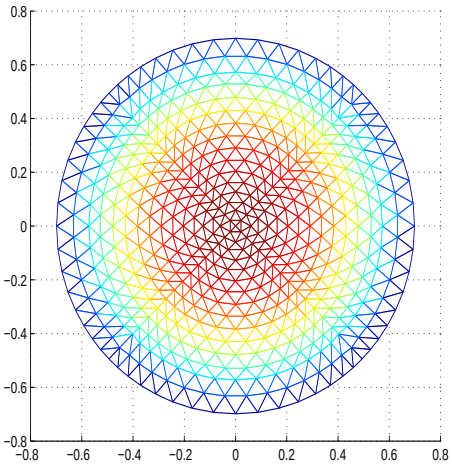

Figure 5.28: Mesh at T=0.1 for 2D run, 545 nodes, $n = 1$.

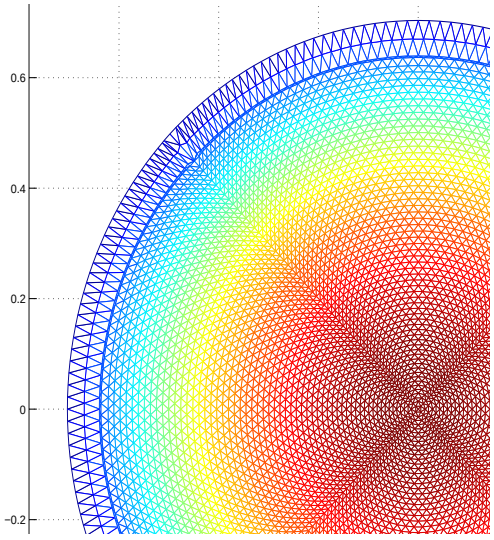

Figure 5.29: Zoomed mesh at T=0.1 for 2D run, 8321 nodes, $n = 1$.

5.7 Summary and Discussion

In the bulk of this chapter we have considered an extension and application of the BHJx algorithm for the arc-length monitor in one space dimension. In order to achieve this, as in the previous chapter, the normal boundary velocity $\dot{\xi} \cdot \hat{\mathbf{n}}$ is determined by using the BHJ approach using a mass monitor. Furthermore, the derivation of the algorithm has been

extended from that for a general monitor function $m(u)$ to one involving a monitor $m(\frac{\partial u}{\partial x})$, depending upon the first spatial derivative of u. In testing the performance of the resulting algorithm, two distinctive implementations were considered. The first of these, known as ALE, was based upon the standard ALE approach, once $\dot{\mathbf{x}}$ had been obtained. The second, known as ALE+, uses the ALE calculation of u as a first estimate in an algorithm that then attempts to find new values of u, that actually force the distribution to be maintained (keep c_i constant).

For the n=3 cases, there was a general lack of robustness, to the point where there was insufficient data to estimate orders of convergence. Only in the ALE+ case with unoptimised data can we show an order of convergence, though that is comparable to the original BHJ values [4]. It may be that the steep slopes for $n = 3$ at the boundary are causing the robustness problems. This is a known problem with the PME [93], but nonetheless we have to say the BHJx algorithm did not work for this case.

For the n=1 cases, initial results showed the importance of imposing the normal boundary velocity $\dot{\xi} \cdot \hat{\mathbf{n}}$ strongly, and so this approach, as outlined in Section 5.3.3, was adopted throughout. Furthermore, the application of the boundary condition $u = 0$ strongly was also found to be superior to only enforcing it weakly. A consequence of using a monitor function of the form $m(\frac{\partial u}{\partial x})$ is that it becomes necessary to estimate an integral of a third derivative of u in the resulting method. Our approach to overcoming this is based upon the projection of $\frac{\partial^2 u}{\partial x^2}$ onto the space of piecewise linear functions, and then applying integration by parts. This was found to be superior to calculating the cubic spline interpolant of u and then taking its third derivative. We also saw (in Section 5.3.3.1) that accuracy was improved by having initial conditions optimised (so we start with an equidistribution of the arc-length monitor), to the point where the order of convergence in the solution error is comparable to that in the original BHJ algorithm [4].

For the n=1 cases, we saw that the ALE+ approach can be beneficial in terms of accuracy, provided that the constraint that the c_i remains constant is not imposed too strictly. Indeed,

with unoptimised initial conditions (an equally-spaced grid), third order accuracy was achieved for the solution error. With optimised initial conditions, we see it is possible to strictly enforce the c_i being constant (and so equidistributing the monitor), but this has cost us an order of accuracy. We can also view this last result as a quantification of accuracy gain or loss, if we allow a distribution to move away from an equidistribution.

Chapter 6

Conclusions

We present four aspects of the results in this thesis, though we do not claim these to be exhaustive, and we then consider the matter of possible future work.

6.1 Providing a general numerical technique

In the guidelines set out in the introduction of this thesis, we have sought to provide a general numerical technique, rather than solving a specific class of mathematical models. It is for this reason that the PDEs have been studied in a simplified, canonical form, as discussed in the introduction. In fact, BHJx has been assessed for two different parabolic PDEs (semilinear heat equation and porous medium equation), three different monitor functions (power, area and arc-length), and in 1D and 2D (though not all combinations of these three factors), and has been tested against meshes that have been untangled, and then restarted (in the blow-up study). We also note that in the 2D PME chapter, the algorithm has been assessed against non (radially) symmetric meshes and in the arc-length chapter, against unoptimised and optimised initial conditions, to ensure the algorithm's basic functionality is not dependent on just one set of initial conditions. Over all these criteria, the BHJx algorithm has been robust enough to allow a study of its accuracy, with the one exception in the arc-length study, where the porous medium equation parameter

n (as in $\nabla(u^n \nabla u)$) was 3.

Let us now scrutinize the changes made to the algorithm throughout the numerical study as well as its definition. In the definition of the algorithm in Section 2.1, we have stated that the velocity potential is taken to be zero on all boundary nodes and that boundary conditions are enforced strongly. For the former condition, this is the same as the assumption made in the original BHJ study [4], when applied to the porous medium equation in 2D, and is equivalent to requiring that the tangential boundary velocity is zero (which is trivially true for our study of the semilinear heat equation, where the boundary is fixed) and is, in any case, very easily changed in the algorithm implementation (this might be required in 1D, if we have non-symmetric initial conditions). For the latter, it was shown in Chapter 5 that BHJx did not function there with weak boundary conditions, but again it is a straightforward change to make in such an implementation, if ever needed.

A significant feature of the algorithm is the need to prescribe a normal mesh velocity at the domain boundary. Where this is not available explicitly it is estimated here by using a *second* monitor function. We have assessed this for the porous medium equation with mass as the second monitor, and with both area and arc-length for the first (interior) monitor. However, the algorithm could (theoretically) be used with any monitor to prescribe the normal boundary velocity, provided it has a first derivative - the calculation (in Section 4.1, for example) is shown for the second monitor being u, but the logic is the same for a more general monitor. We have also imposed no conditions on the main (interior) monitor $m(u)$, save that it has a first derivative[1]. This assumption also applies to monitor functions of the form $m(v)$, where $v = \frac{\partial u}{\partial x}$, as studied in Chapter 5, and it is worth noting that the method there is provided for a general $m(v)$, even though we have only assessed it in 1D for the arc-length monitor, where $v = \sqrt{(}1+v^2)$. We should point out here that the algorithm then has a slightly different form, as we need to estimate third order spatial derivatives.

In the arc-length chapter, where we have attempted to force the c_i to be constant, in

the "ALE+" method, we do have, as things stand, a parameter that depends on initial conditions and mesh refinement - the SUNDIAL tolerance. In particular, this parameter requires different values for unoptimised and optimised initial conditions. We would argue that this parameter could be amended to be adjustable by the algorithm, so that if it is required to maintain a distribution (keep the c_i constant), the SUNDIAL tolerance will be reduced from its default value until this is achieved. We concede though, that this may not always be possible, i.e., the SUNDIAL suite may not converge, so we have then provided a general method, but may have lost robustness in the process of doing so, from the point of view of supplying an algorithm that *must* maintain the distribution. However, we have shown in the arc-length study, *and* the 2D-PME study, that the monitors there can do what might be expected of them - attempt to conserve their distribution, at least in the long term.

6.2 Comparison with other techniques and monitors

The main performance criteria in this thesis have been accuracy and robustness, and it will be these that we use here to compare BHJx with other techniques. In the case of the porous medium equation, since we showed in the foundation study in Section 2.2 that BHJx had comparable orders of accuracy with BHJ for the mass monitor, this is effectively a comparison of other monitors with the mass monitor.

We will review the thesis results from the aspect of accuracy and robustness, in the order in which they appear. In Chapter 3, studying the blow-up problem, we have seen second order accuracy (for an extrapolated result), but robustness has been a problem, mostly due to tangling, and this tangling being mostly near to the boundary. The fact that u as a monitor allowed a higher u_{max} to be reached than u^2 for the $p = 3$ case (the PDE is $u_t = \triangle u + u^p$) agrees with findings by Twigger [91], who also uses a velocity based

[1] To simplify some of the calculus in some cases, we have assumed that $m(u)$ has a second derivative.

adaptive method, but in general, BHJx has not reached as high a value of u_{max} as the study by Budd, Huang and Russell [20]. There has been agreement with [20] on the blow-up happening at a single point x^* (which is at the peak of the initial data, as found in both [20] and [17]), and the blow-up time T has been comparable with the study in [20]. In fact, since we have basic agreement on x^* and T, if we then compare only with [20], that leaves us with looking at the highest value of u_{max} that can be reached before the algorithm breaks down. This value has not been as high as that in [20], though it was getting higher as the mesh was refined, and also after tangled meshes were untangled, and the run restarted. We see then, that BHJx has second order accuracy here, and has some robustness, but is not as robust as [20], mostly due to tangling.

In Chapter 4, we studied the area monitor in 2D, applied to the porous medium equation. Here we found that the area monitor $u + a$ was less accurate than the mass monitor as a increases, but can be more robust for non (radially) symmetric meshes. This robustness appears to be due to the intrinsic nature of the area monitor, in that it seeks to maintains an area distribution - this point will be explored more closely in Section 6.4.

Finally, in Chapter 5, we studied the arc-length monitor in 1D, applied to the porous medium equation. With the algorithm applied in the usual way ("ALE"), but with the adjustment of the normal boundary velocity being applied directly, we see robustness, and for optimised initial conditions, an order of accuracy comparable to the original BHJ study [4], though all this is only for $n = 1$. In the case $n = 3$ there is not even enough robustness to measure any accuracy! In the special cases where we force the distribution (so c_i) to be constant ("ALE+"), we achieved order 3 accuracy in the solution for unoptimised initial conditions, just by *attempting* to keep c_i constant, though this was not robust when the c_i were actually forced to be constant. For optimised initial conditions and ALE+, we were able to maintain the distribution *and* achieve robustness, though this reduced the order of accuracy to 1 (from 2 obtained in BHJ [4]). This last result is directly comparable to BHJ, as that also forces the c_i to be constant - this forcing *is* its method of calculating u (see

Section 1.8).

Generally, we can see here that the BHJx method has had some robustness problems in 1D, and in both 1D and 2D, the area and arc-length monitor have not been as accurate as the mass monitor, when studying the porous medium equation. There have however, been two notable exceptions to this statement:-

- In 1D, with unoptimised initial conditions, the arc-length monitor was able to achieve third order accuracy, by attempting to maintain the monitor distribution.
- In 2D, for the porous medium equation, the area monitor can be more robust than the mass monitor for some domains, as it can prevent (for a while) elements shrinking to the point of disappearing.

6.2.1 Comparison specifically with BHJ

We note from Section 6.2 that BHJx has generally been less accurate and robust than BHJ when using area and arc-length monitors, with the most notable exception being the area monitor in Chapter 4, where BHJx was more robust. We can also state here that the CPU time for BHJx in Chapter 4 was generally twice that of BHJ, and this can mostly be attributed to derivation of the normal boundary velocity, as this is effectively repeating a large part of the algorithm with the mass monitor. Further than this, in the arc-length study in Chapter 5, the "ALE+" version did, in the worst case, take up to four times as much CPU time as the "ALE" version, due to the extra constraint of forcing the c_i to be constant. We can say generally then, that as it stands, BHJx can take at least twice the CPU time of BHJ, due to the derivation of boundary velocities, and is less accurate and robust, except for the area monitor in 2D, which showed more robustness than BHJ.

6.3 Tangling issues

There have been tangling problems with the BHJx algorithm, which may be due to its very imposition of trying to maintain a distribution. It has been stated in the 2009 review paper by Budd, Huang and Russell [15] that velocity-based methods (such as BHJx) are more prone to tangling than location-based methods (such as MMPDE), and this is certainly borne out by the blow-up study in Chapter 3. In that chapter, the tangling was overcome to some degree by untangling and restarting meshes. This was done manually, but it would be feasible to incorporate this untangling and restarting within the algorithm, leading to a more robust method, and allowing a higher u_{max} to be reached. In that sense, we could say that tangling has not caused a problem, though there will be a limit to how much untangling can be done, as we saw in the case for the monitor function $m(u) = u$ and $p = 3$, which would not restart at all. A better approach for this problem, in the blow-up modelling, might be to have some form of smoothing or regularity applied, as in general, reducing the timestep did not prevent the tangling problem. In Chapter 4, we saw the area monitor was actually less prone to tangling than the mass monitor. Furthermore, the area monitor by itself can be used to prevent (or at least postpone) tangling, which has the advantage of needing no extra mesh management, though some accuracy may be lost. For the arc-length monitor, tangling has not really been an issue for $n = 1$, but for $n = 3$, it has been a problem, though the BHJx algorithm itself broke down in some cases here before tangling (potentially) occured, so it is not clear whether even untangling or applying smoothing or regularisation would help here.

In general then, we see that there have been tangling problems with BHJx, though they can be surmounted, and using one monitor, the area monitor, can actually attenuate these problems by its very nature.

6.4 Knowledge gained beyond the BHJx assessment

The purpose of this thesis is to introduce and assess the BHJx algorithm. However, there are some other, more general results we can find here. In Chapter 4, we saw that the area monitor could be used to model the porous medium equation more robustly than the mass monitor for a non (radially) symmetric domain. A qualitative analysis showed that both monitors, at least in the long-term, were conserving their distribution, as we expected them to. Thus the question of what effect maintaining a distribution has on robustness has been partly answered - we can say in the case mentioned above, that if the area monitor does maintain its distribution, this can give us increased robustness. On the other hand, we can view the results in the arc-length chapter ("ALE+" case), as giving us information on what happens if a mesh moves away from an equidistribution - we saw there that, looking at those results in the opposite order, moving away from an equidistribution has gained us an order of accuracy. So although the aim of a monitor function is to maintain an initial distribution (an equidistribution for optimised initial conditions), we do not necessarily achieve the best accuracy if we achieve that aim, conversely the result with unoptimised initial conditions show we can improve accuracy by just *attempting* to maintain a distribution. We also note that using *two* monitors has been far more successful than one - this is one of the key features of the BHJx algorithm.

6.5 Future work

There was an open question raised in Section 6.4 on how to quantify the effects of maintaining (or moving away from) an (equi)distribution, on accuracy and robustness, and it would be worth using BHJx to explore this further. The first port of call here would be to look more closely at the arc-length results, particularly where there was third-order accuracy, which was when the initial conditions were unoptimised for the arc-length monitor (so the grid was just equally spaced) in the ALE+ case (Section 5.4.2). This needs

scrutinising, not just because it is a positive result, but because we have to ask ourselves if it was the attempt to force the c_i to be constant that actually improved the order, and then why did actually forcing the c_i to be constant essentially fail (the failure being in the SUNDIAL suite). It might be that the attempt to force the c_i to be constant moved the evolution of the c_i closer to an idealised evolution for the porous medium equation (which is simple enough to define for the central and boundary nodes, using the known solution), and so produced a more accurate result.

In the first place though, more refinement is needed to see if the third order accuracy holds, as it is only being seen over three points in the log-log graph (Figure 5.14). When the initial conditions were optimised (in Section 5.4.3), the c_i did remain constant, but an order of accuracy was lost, so if we assume that the c_i remaining constant correlates with losing accuracy, then as the c_i remain constant for the 11-node case with unoptimised initial conditions, but not for higher refinements, we might expect that the third order accuracy would continue as the grid is refined.

Secondly, as the accuracy results are taken at $T = 1.0$, but we can clearly see the evolution of c_i settling down at that time, it would be worth repeating these runs at some smaller times, to see what the accuracy is there. Further than that, runs could be done with another PDE, with a known solution, such as the Oxygen problem in 1D (see Section 2.2). We could also try the area monitor, for the Oxygen problem in 1D (and 2D), to further investigate robustness effects, when a distribution is maintained.

Another extension could be to monitors with second or higher order spatial derivatives, such as a curvature monitor. This will ultimately require estimates of these higher order derivatives, which can be made by a weak formulation - a process similar to finding $\frac{\partial(Lu)}{\partial x}$ in Chapter 5, though the process will need to be repeated (see [4] for example, on dealing with fourth order PDEs).

Considering a more general extension, this thesis has assessed the BHJx algorithm for parabolic problems, so a possible next stage is to assess it for hyperbolic problems. For

example, a study could be made of the non-linear Schrödinger equation [64, 67, 86, 91]. One form of this is:-

$$-ihu_t = \frac{h^2}{2m}\triangle u + U(u^2)u, \tag{6.1}$$

where h is Planck's constant, m is mass and U represents non-linear effects. The dependent variable is a complex number, but can be found in practice by representing as $u = v + iw$, or by separating variables. As with the semilinear heat equation, studied in Chapter 3, we could use a fixed boundary, and so prescribe the normal boundary velocity as zero. Equation (6.1) was originally used by Schrödinger to correctly predict frequencies of the Hydrogen atom [43]. He did not originally prescribe u as having a physical meaning, but it was later thought of as an "essence", and later as giving rise to a probability density of a particle (or other physical measurement) being in a particular region of space. From a computational and scientific point of view then, we can see it as a non-linear hyperbolic PDE, though its abstract nature makes it less obvious what a monitor function should be, from a conservative or distribution point of view. However, from a computational point of view, using scale-invariance, Budd and Piggott [16] have used the monitor function $m(u) = |u|^2$. In fact, if $U(u^2) =$constant in 6.1, the probability of finding a particle in a region Ω of space is $\int_\Omega |u|^2 d\Omega$, therefore $\int_{\Re^d} |u|^2 d\Omega$ must be 1, which does mean we have an overall "conservation of probability" to 1.

Another form, the radially symmetric non-linear Schrödinger equation, is:-

$$-iu_t = u_{xx} + \frac{d-1}{x}u_x + u|u|^2, \tag{6.2}$$

where d is the dimension of the domain and x is the distance from the origin. It is used to model water waves and plasma waves [16]. We can see that in 1D, it reduces to the first form. The first form has known similarity solutions and a global solution, but the second, for 2D and higher has a blow-up problem [16] at the origin.

The BHJx method could be used to study the first form, where we can study robustness and accuracy items. For 2D, we can focus on providing an alternative method to Budd and Piggott [16] of modelling the blow-up problem.

Bibliography

[1] Adams, R. A. and Fournier, J. J. F. *Sobolev Spaces (second edition)*. Academic Press, 2003.

[2] Allaire, G. and Brizzi, R. A multiscale finite element method for numerical homogenization. *Multiscale Modeling and Simulation*, 4(3):790–812, 2006.

[3] Apostol, T. M. *Mathematical analysis (second edition)*. Addison Wesley, 1974.

[4] Baines, M. J., Hubbard, M. E. and Jimack, P. K. A moving mesh finite element algorithm for the adaptive solution of time-dependent partial differential equations with moving boundaries. *Applied Numerical Mathematics*, 54:450–469, 2005.

[5] Baines, M. J., Hubbard, M. E., Jimack, P. K. and Jones, A. C. Scale-invariant moving finite elements for nonlinear partial differential equations in two dimensions. *Applied Numerical Mathematics*, 56:230–252, 2006.

[6] Bangerth, W. and Rannacher, R. *Adaptive finite element methods for differential equations*. BirkHauser, 2003.

[7] Bänsch, E. An adaptive finite-element strategy for the three-dimensional time-dependent Navier-Stokes equations. *Journal of Computational and Applied Mathematics*, 36(1):3–28, 1991.

[8] Berger, A. E. and Ciment, M. and Rogers, J. C. W. Numerical solution of a diffusion consumption problem with a free boundary. *SIAM Journal on Numerical Analysis*, 12(4):646–672, 1975.

[9] Berzins, M., Capon, P. J. and Jimack, P. K. On spatial adaptivity and interpolation when using the method of lines. *Applied Numerical Mathematics*, 26(1):117–134, 1998.

[10] Berzins, M., Fairlie, R., Pennington, S. V., Ware, J. M. and Scales, L. E. Sprint2d: Adaptive software for PDEs. *ACM Transactions on Mathematical Software*, 24(4):475–499, December 1998.

[11] Brenner, S. C. and Scott, L. R. *The Mathematical Theory of Finite Element Methods (second edition)*. Springer, 2002.

[12] Briggs, W. L., Henson, V. E. and McCormick, S. F. *A Multigrid Tutorial (second edition)*. Siam, 2000.

[13] Brown, P. N. A local convergence theory for combined inexact-Newton / finite-difference projection methods. *SIAM Journal on Numerical Analysis*, 24(2):407–434, 1987.

[14] Budd, C. J. and Collins, G. J. An invariant moving mesh scheme for the nonlinear diffusion equation. *Applied Numerical Mathematics*, 26:23–39, 1998.

[15] Budd, C. J. and Huang, W. and Russell, R. D. Adaptivity with moving grids. *Acta Numerica*, pages 1–131, 2009.

[16] Budd, C. J. and Piggott, M. D. The geometric integration of scale-invariant ordinary and partial differential equations. *Journal of Computational and Applied Mathematics*, 128:399–422, 2001.

[17] Budd, C. J. and Williams, J. F. Parabolic Monge-Ampère methods for blow-up problems in several spatial dimensions. *Journal of Physics - A Mathematical and General*, 39(19):5425–5444, May 2006.

[18] Budd, C. J. and Williams, J. F. Moving mesh generation using the Parabolic Monge-Ampère equation. *SIAM Journal on Scientific Computing*, 31:3438, 2009.

[19] Budd, C. J. and Williams, J. F. How to adaptively resolve evolutionary singularities in differential equations with symmetry. *Journal of Engineering Mathematics*, 66:217–236, 2010.

[20] Budd, C. J., Huang, W., and Russell, R. D. Moving mesh methods for problems with blow-up. *SIAM Journal on Scientific Computing*, 17(2):305–327, 1996.

[21] Burman, E., Jacot, A. and Picasso, M. Adaptive finite elements with high aspect ratio for the computation of coalescence using a phase-field model. *Journal of Computational Physics*, 195:153–174, 2004.

[22] Cai, X. X., Fleitas, D., Jiang, B., and Liao, G. Adaptive grid generation based on Least-squares finite element method. *Computers and Mathematics with applications*, 48:1077–1086, 2004.

[23] Cao, W. and Demkowicz, L. Optimal error estimate of a projection based interpolation for the p-version approximation in three dimensions. *Computers & Mathematics with Applications*, 50(3-4):359–366, 2005.

[24] Cao, W. M., Huang, W. Z. and Russell, R. D. A moving mesh method based on the geometric conservation law. *SIAM Journal on Scientific Computing*, 24(1):118–142, August 2002.

[25] Carey, G. F. and Oden, J. T. *Finite Elements (Fluid Mechanics) Volume VI*. Prentice-Hall, 1986.

[26] Ceniceros, H. D. and Hou, T. Y. An efficient dynamically adaptive mesh for potentially singular solutions. *Journal of Computational Physics*, 172(2):609–639, 2001.

[27] Chari, M. V. K. and Silvester, P. P. *Finite Elements in Electrical and Magnetic Field Problems*. Wiley, 1980.

[28] Cole, S. L. Blow-up in a Chemotaxis Model Using a Moving Mesh Method. Master's thesis, Department of Mathematics, University of Reading, 2009.

[29] Davis, S. F. and Flaherty, J. E. An adaptive finite element method for initial-boundary value problems for partial differential equations. *SIAM Journal on Scientific and Statistical Computing*, 3(1), March 1982.

[30] Delzanno, G. L. and Chacón, L., Finn, J. M., Chung, Y. and Lapenta, G. An optimal robust equidistribution method for two-dimensional grid adaptation based on Monge-Kantorovich optimization. *Journal of Computational Physics*, 227(23):9841–9864, 2008.

[31] Di, Y. N., Li, R., Tang, T. et al. Moving mesh methods for singular problems on a sphere using perturbed harmonic mappings. *SIAM Journal on Scientific Computing*, 28(4):1490–1508, 2006.

[32] Dirac, P.A.M. *The principles of quantum mechanics*. Oxford University Press, USA, 1981.

[33] Dorfi, E. A. and Drury, L. O' C. Simple adaptive grids for 1-D initial value problems. *Journal of Computational Physics*, 69(1):175–195, 1987.

[34] Elman, H., Silvester, D. and Wathen, A. *Finite Elements and Fast Iterative Solvers*. OUP, 2005.

[35] Étienne, S., Garon, A. and Pelletier, D. Perspective on the geometric conservation law and finite element methods for ale simulations of incompressible flow. *Journal of Computational Physics*, 2008.

[36] Farhat, C., Geuzaine, P. and Grandmont, C. The discrete geometric conservation law and the nonlinear stability of ALE schemes for the solution of flow problems on moving grids. *Journal of Computational Physics*, 174(2):669–694, 2001.

[37] Filo, J. and Pluschke, V. The porous medium equation in a two-component domain. *Journal of Differential Equations*, 247(9):2455–2484, 2009.

[38] Gaskell, P. H., Koh, Y. Y., Jimack, P. K., Sellier, M., Thompson, H. M. and Wilson, M. C. T. Thin film flow over substrates with topography. In *12th International Coating Science and Technology Symposium*. Citeseer, 2004.

[39] Gatica, G. N., Gonzalez, M. and Meddahi, S. A low-order mixed finite element method for a class of quasi-Newtonian Stokes flows. Part II: a posteriori error analysis. *Computer Methods In Applied Mechanics And Engineering*, 193:893–911, 2004.

[40] Gilbarg, D. and Trudinger, N. S. *Elliptic Partial Differential Equations of Second Order*. Springer-Verlag, 1983.

[41] Hawken, D. F., Gottlieb, J. J. and Hansen, J. S. Review of some adaptive node-movement techniques in finite-element and finite-difference solutions of partial differential equations. *Journal of Computational Physics*, 95:254–302, 1991.

[42] Hindmarsh, A. C., Brown, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E. and Woodward, C. S. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):396, 2005.

[43] Hoffmann, B. *The Strange Story of the Quantum*. Penguin, 1963.

[44] Huang, W. and Russell, R. D. A moving collocation method for solving time dependent partial differential equations. *Applied Numerical Mathematics*, 20(1-2):101–116, 1996.

[45] Huang, W. and Russell, R. D. A high dimensional moving mesh strategy. *Appl. Numer. Math*, 26:63–76, 1997.

[46] Huang, W. and Russell, R. D. Moving mesh strategy based on a gradient flow equation for two-dimensional problems. *SIAM J. SCI. COMPUT*, 20(3):998–1015, 1998.

[47] Huang, W. and Russell, R. D. Adaptive mesh movement - the MMPDE approach and its applications. *Journal of Computational and Applied Mathematics*, 128(1-2):383–398, 2001.

[48] Huang, W., Ma, J. and Russell, R. D. A study of moving mesh PDE methods for numerical simulation of blow-up in reaction diffusion equations. *Journal of Computational Physics*, 227(13):6532–6552, 2008.

[49] Huang, W., Ren, Y. and Russell, R. D. Moving mesh methods based on moving mesh partial differential equations. *J. Comput. Phys*, 113(2):279–290, 1994.

[50] Huang, W., Ren, Y. and Russell, R. D. Moving mesh partial differential equations (MMPDES) based on the equidistribution principle. *SIAM Journal on Numerical Analysis*, 31(3):709–730, 1994.

[51] Hubbard, M. E., Baines, M. J. and Jimack, P. K. Consistent Dirichlet boundary conditions for numerical solution of moving boundary problems. *Applied Numerical Mathematics*, 59(6):1337–1353, 2009.

[52] Hughes, W. F. and Brighton, J. A. *Schaum's Outline of Theory and Problems of Fluid Dynamics*. McGraw-Hill, 1967.

[53] Jacques, I. and Judd, C. *Numerical analysis*. Chapman and Hall, 1987.

[54] Jeffreys, H. and Swirles, B. *Methods of mathematical physics*. Cambridge Univ Pr, 1999.

[55] Jimack, P. K. A best approximation property of the moving finite element method. *SIAM J. Numer. Anal*, 33(6):2286–2302, 1996.

[56] Jones, A. C. and Jimack, P. K. An adaptive multigrid tool for elliptic and parabolic systems. *International Journal for Numerical Methods in Fluids*, 47(10-11):1123–1128, 2005.

[57] Khassehkhan, H. and Eberl, H. J. Interface tracking for a non-linear, degenerated diffusion-reaction equation describing formation of bacterial biofilms. *Dynamics of Continuous Discrete and Impulsive Systems Series A-Mathematical Analysis*, 13(6):131–144, Oct 2006.

[58] Khassehkhan, H. and Eberl, H. J. Modeling and simulation of a bacterial biofilm that is controlled by pH and protonated lactic acids. *Computational and Mathematical Methods in Medicine*, 9(1):47–67, 2008.

[59] Kinnunen, J. and Lindqvist, P. Definition and properties of supersolutions to the porous medium equation. *Journal fur die reine und angewandte Mathematik*, 618:135–168, 2008.

[60] Korn, G. A. and Korn, T. M. *Mathematical Handbook for Scientists and Engineers: Definitions, Theorems, and Formulas for Reference and Review*. Courier Dover Publications, 2000.

[61] Lambert, J. D. *Computational Methods in Ordinary Differential Equations.* Wiley, London, 1973.

[62] Lankalapalli, S., Flaherty, J. E., Shephard, M. S. and Strauss, H. An adaptive finite element method for magnetohydrodynamics. *Journal of Computational Physics*, 225(1):363–381, 2007.

[63] Liao, G. and Anderson, D. A new approach to grid generation. *Applicable analysis(Print)*, 44(3-4):285–298, 1992.

[64] Lin, J. E. and Strauss, W. A. Decay and scattering of solutions of a nonlinear Schrödinger equation. *Journal of Functional Analysis*, 30(2):245–263, 1978.

[65] Mackenzie, J. A. and Robertson, M. L. A moving mesh method for the solution of the one-dimensional phase-field equations. *Journal of Computational Physics*, 181(2), 2002.

[66] B. B. Mandelbrot. *The fractal geometry of nature*. Freeman, 1983.

[67] Markowich, P. A., Pietra, P. and Pohl, C. Numerical approximation of quadratic observables of Schrödinger-type equations in the semi-classical limit. *Numerische Mathematik*, 81(4):595–630, 1999.

[68] Maron, M. J. *Numerical Analysis - a Practical Approach (second edition)*. Macmillan, 1987.

[69] McCarthy, J. F. One-dimensional phase-field models with adaptive grids. *ASME Transactions*, 120, Nov 1998.

[70] Melenk, J. M. and Babuška, I. The partition of unity finite element method: basic theory and applications. *Computer methods in applied mechanics and engineering*, 139(1-4):289–314, 1996.

[71] Miller, K. Moving finite elements. II. *SIAM Journal on Numerical Analysis*, 18:1033–1057, 1981.

[72] Miller, K. and Miller, R. N. Moving Finite Elements. I. *SIAM Journal on Numerical Analysis*, 18:1019–1032, 1981.

[73] Montijn, C., Hundsdorfer, W. and Ebert, U. An adaptive grid refinement strategy for the simulation of negative streamers. *Journal of Computational Physics*, 219(2):801–835, 2006.

[74] Murray, J. D. *Mathematical biology*. Springer Verlag, 2003.

[75] Olver, P. J. *Applications of Lie Groups to Differential Equations*. Springer-Verlag, 1983.

[76] Pepper, D. W. and Heinrich, J. C. *The Finite Element Method*. Taylor and Francis, 1992.

[77] Piggott, M. D., Farrell, P. E., Wilson, C. R., Gorman, G. J. and Pain, C. C. Anisotropic mesh adaptivity for multi-scale ocean modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1907):4591, 2009.

[78] Plewa, T., Linde, T. and Weirs, V. G. *Adaptive Mesh Refinement - Theory and Applications*. Springer, 2004.

[79] Reddy, J. N. *An Introduction to the Finite Element Method*. McGraw-Hill, 1984.

[80] Remacle, J. F., Flaherty, J. E. and Shephard, M. S. An adaptive discontinuous Galerkin technique with an orthogonal basis applied to compressible flow problems. *SIAM review*, 45(1):53–72, 2003.

[81] Ren, Y. and Russell, R. D. Moving mesh techniques based upon equidistribution, and their stability. *SIAM Journal on Scientific and Statistical Computing*, 13(6):1265–1286, Nov 1992.

[82] Rivière, B. *Discontinuous Galerkin methods for solving elliptic and parabolic equations: theory and implementation*. Society for Industrial & Applied Mathematics, 2008.

[83] Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computation, 1994.

[84] Speares, W. and Berzins, M. A 3-d unstructured mesh adaption algorithm for time-dependent shock dominated problems. *International Journal for Numerical Methods in Fluids*, 25(81):104, 1997.

[85] Strang, G. and Fix, G. J. *An Analysis of the Finite Element Method*. Prentice-Hall, 1973.

[86] Strauss, W. A. *Partial differential equations: An introduction*. John Wiley & Sons New York, 2008.

[87] Tabarraei, A. and Sukumar, N. Adaptive computations on conforming quadtree meshes. *Finite Elements in Analysis & Design*, 41(7-8):686–702, 2005.

[88] Tang, H. and Tang, T. Adaptive mesh methods for one-and two-dimensional hyperbolic conservation laws. *SIAM Journal on Numerical Analysis*, 41(2):487–515, 2003.

[89] Thomas, P. D. and Lombard, C. K. Geometric conservation law and its application to flow computations on moving grids. *AIAA Journal*, 17(10):1030–1037, 1979.

[90] Thompson, J. F., Soni, B. K. and Weatherill, N. P. *Handbook of grid generation*. CRC, 1999.

[91] Twigger, A. Blow-up in the Nonlinear Schrödinger Equation Using an Adaptive Mesh Method. Master's thesis, Department of Mathematics, University of Reading, 2008.

[92] Vazquez, J. L. *The porous medium equation: Mathematical Theory*. Oxford University Press, USA, 2007.

[93] Wacher, A. and Sobey, I. String gradient weighted moving finite elements in multiple dimensions with applications in two dimensions. *SIAM Journal on Scientific Computing*, 29(2):459–480, 2007.

[94] Weissler, F. B. Existence and non-existence of global solutions for a semilinear heat equation. *Israel Journal of Mathematics*, 38(1):29–40, 1981.

[95] White, A. B. Jr. On selection of equidistributing meshes for two-point boundary-value problems. *SIAM Journal on Numerical Analysis*, 16(3):472–502, 1979.

[96] Wilcox, L. C., Stadlera, G., Bursteddea, C. and Ghattasa, O. A high-order discontinuous Galerkin method for wave propagation through coupled elastic-acoustic media. *Submitted to Journal of Computational Physics, doi: 10.1016/j.jcp.2010.09.008*, 2010.

[97] Williams, W. E. *Partial Differential Equations*. Oxford University Press, 1980.

[98] Zegeling, P. A. and Kok, H. P. Adaptive moving mesh computations for reaction-diffusion systems. *Journal of Computational and Applied Mathematics*, 168(1-2):519–528, Jul 2004.