A Projected Multigrid Method for the Solution of Nonlinear Finite Element Problems on Adaptively Refined Grids

by

Alison Claire Jones

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

The University of Leeds
School of Computing

Submitted: July 2005

The candidate confirms that the work submitted is her own and the appropriate credit has been given where reference has been made to the work of others.

Abstract

This thesis describes the formulation and application of an adaptive multigrid method for the efficient solution of nonlinear elliptic and parabolic partial differential equations and systems. A continuous Galerkin finite-element method is combined with locally adaptive mesh refinement and an optimal multigrid solver to achieve this efficiency. The novel contribution of this work lies in the manner in which these two techniques are combined. In particular the multigrid solver provides a natural and simple method of handling grid points that are not fully connected, so called *hanging nodes*. This allows for a straightforward adaptive gridding scheme that does not need to take any special measures to repair these hanging nodes for a standard element-by-element implementation of the finite element assembly process. Specifically, on each element, only the usual finite element basis functions are required, even in the vicinity of hanging nodes. Furthermore the standard multigrid full approximation scheme (FAS) may be applied with only minor modifications to account for the presence of the hanging nodes. A wide cross-section of nonlinear elliptic and parabolic problems are used to demonstrate the performance of the proposed algorithm, which is shown to provide optimal accuracy at an optimal computational cost.

Acknowledgements

I would like to thank: Peter Jimack for all of his teaching, feedback, support, encouragement and incredible unwavering belief in me; Mark Walkley for giving up his time and expertise to help; Chris Goodyer for his patience in the face of many daft questions; Andrew Mullis for sharing his knowledge of phase-field models and Daniel Hart for ‘it’s nearly Christmas after all’.

Thanks are also due to: Peter Jones and Kate Livingston for being brave enough to read my paper; Margaret Jones and Lorna Foott for listening without a chance of understanding; Joseph Casey for holding the pieces together; Timothy Peffers, Christina Dunning and Benjamin Howe for putting up with my grumpiness and putting a smirk back on my face and Clare MacIver for patiently waiting a year for me to stop working and visit her.

Declarations

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

A. C. Jones and P.K. Jimack, “An adaptive multigrid tool for elliptic and parabolic systems”, *International Journal for Numerical Method in Fluids*, 47 (10-11) pp1123-1128, 2005.

M.J. Baines, M.E. Hubbard, P.K. Jimack and A. C. Jones, “Scale-invariant moving finite elements for nonlinear partial differential equations in two dimensions”, *Applied Numerical Mathematics*, to appear.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This thesis is concerned with the efficient numerical solution of a wide range of partial differential equations (PDEs) and systems, of elliptic and parabolic type. In order to obtain such efficiency it is necessary to combine adaptivity in the spatial discretisation with the use of an optimal multigrid solver and, for time-dependent problems, fully implicit, unconditionally stable, time-stepping schemes. The main new contribution of this work is the proposal of a novel method for combining mesh adaptivity with a multigrid solver in a manner that retains the optimal behaviour of the existing schemes but does so via a very natural and simple algorithm. The performance of this proposed technique is assessed across a wide cross-section of computational problems including nonlinear elliptic and parabolic problems.

A multigrid solver, for use with problems defined over locally refined grids, and a complementary gridding scheme are both developed within this work. The h-adaptive gridding scheme allows for aggressive local refinement, as well as dynamic refinement and de-

refinement. This is of particular use in solving problems containing moving fronts, or shocks, which require high grid resolution. The adaptivity can be driven by any error estimator (or indicator) which produces a value for each element in the grid. A hierarchical data structure is used to store the grid elements giving a natural framework for the local h-adaptation. We chose to allow disconnected (or 'hanging') nodes in the resulting non-uniform grids to avoid extra refinement. In order to perform a continuous Galerkin finite element solve over a mesh containing hanging nodes something must be done to resolve the potential discontinuities created by the hanging nodes along the edges where they lie. A Projected FAS (PFAS) multigrid solver is presented, which naturally resolves the solution values at these points. The smoothing steps of this PFAS scheme are performed using a nonlinear Jacobi iterative smoother. This smoother makes use of a projection technique first introduced by Meyer [67] to resolve the hanging node values. The FAS scheme itself, although largely unchanged by the use of the new smoother, must make use of specific restriction and prolongation schemes.

The performance of the PFAS scheme is assessed for a nonlinear elliptic model problem and three nonlinear parabolic problems. The three parabolic problems are the porous medium equation, a system of Burgers' equations and a phase field (PF) model of rapid solidification. The specific PF model [97] used represents the solidification of a pure metal, from an undercooled melt, in a manner which avoids the need to track the interface between the solid and liquid areas. This is an example of a class of problems with much current interest in materials science and potential industrial significance [69].

1.1 Overview

The first two chapters of this thesis present some background to the work. Chapter 2 introduces the numerical methods that are used throughout this work and Chapter 3 gives

a summary of the problems used to illustrate the performance of our proposed method. The h-adaptive gridding scheme is presented in Chapter 4. The chosen data structures and search algorithms are described along with the algorithms for refinement and coarsening of a quadrilateral mesh. A semi-implicit discretisation of a PF model is solved (using a variant of the Conjugate Gradient method, due to Meyer [67]) in order to demonstrate the robustness of the gridding techniques. Details of the new PFAS multigrid solver are given in Chapter 5. The performance of the new method is shown initially for a nonlinear elliptic model problem, for which an exact solution is known. In Chapter 6 the PFAS method is tested further on two nonlinear parabolic problems, including one system of parabolic equations. Firstly a porous medium equation [29] is solved, with nonlinearity in the diffusion term. Then a system of PDEs based on Burgers' equation [23] is considered. The nonlinearity is within the convection terms in this instance. The PF model from Chapter 4 is revisited in Chapter 7. Here a fully implicit time discretisation is used which, although it allows for larger stable time steps, produces a highly nonlinear algebraic system at each step. Again the performance of the PFAS scheme is shown for this problem. Finally, in Chapter 8 this work is reviewed and conclusions are drawn from the results that have been shown. Several natural extensions to this work are also briefly discussed.

Chapter 2

Numerical Methods Background

2.1 Introduction

This work focuses on the numerical solution of partial differential equations, including elliptic problems such as Poisson's equation, and parabolic problems such as Burgers' equation. Other nonlinear problems are also considered, such as the porous medium equation, and parabolic systems such as those arising in phase field models of rapid crystal growth. More details of these applications are given in Chapter 3. In this chapter we discuss the solution process for these two classes of problem (i.e. elliptic and parabolic) from discretisation techniques through to solution methods. Section 2.2 presents some popular options for spatial and temporal (in the parabolic case) discretisation of the continuous problem. Section 2.3 outlines the possibilities for the solution of the algebraic equations resulting from such discretisations. Since the focus of this research is the development of an adaptive multigrid algorithm a detailed description of multigrid methods is then given in Section 2.4. In Sections 2.5 and 2.6 we discuss adaptive methods and

their combination with multigrid solution techniques respectively.

2.2 Discretisation

In this section we will discuss possible spatial and temporal discretisation methods for elliptic and parabolic problems. In Section 2.2.1 we give the main options for spatial discretisation and detail the finite element method which is used extensively in this work. Temporal discretisation is discussed in Section 2.2.2 where various explicit and implicit schemes that are employed throughout this work are introduced.

2.2.1 Spatial Discretisation

Spatial discretisation is the method employed to convert the original continuous partial differential equation to a system of equations which are discrete with respect to the spatial variables. For an elliptic problem this results in a system of algebraic equations whereas, for a parabolic problem it results in a system of ordinary differential equations in time. There are several, well established, methods for spatial discretisation. The finite difference (FD) method is a point-wise technique which uses interpolation formulae to approximate derivatives at each grid point. Stencils for these interpolations are defined over a few neighbouring points. This method was presented in a mature form as long ago as 1967 by Richtmyer and Morton [78]. More recently, [86], for example, provides an alternative guide to the solution of PDEs using finite differences. The, more recent, finite element (FE) approach is based around the regions (or elements) formed between the edges that join points of the grid. This is a global integration method using integrals over the individual elements to build up a full approximation across the grid. Reddy's book [76] provides a good introductory guide to the FE method as does Zienkiewicz and Taylor's text [99] which sets the method in its original discipline of engineering. For a detailed history of the FD and FE methods see [91]. A third possible approach to spatial

discretisation is the Finite Volume (FV) method. This uses averaged values and volume integrals over the individual elements to approximate the solution. In text [93] the FV method is used as the standard technique for CFD problems. Furthermore, this technique has been used widely in CFD software such as Fluent [37] and CFX [3] for example. The application to non-uniform grids is tackled in [46]: this is more straightforward than applying FD methods but, as we shall see below, not quite as natural as using FE methods.

For this work, the FD method was rejected due to the complexities of implementing adaptively. The derivative approximations become complex where the distances between the points vary and the interpolation stencils are non-uniform. Conversely, the FE method is effective on non-uniform grids as the contribution from each element is scaled by its size. This makes it an attractive method for this work as we are dealing with highly non-uniform grids. For the class of problems solved here the FV method is unnecessarily complicated (although for hyperbolic problems it would be worth considering). In light of this, the method of choice for this work is the Continuous Galerkin FE method. This method is explained in more detail below. Before doing so however we briefly note other possible spatial discretisation techniques not considered here. These include collocation methods [56], spectral methods [41] and discontinuous Galerkin methods [31].

Continuous Galerkin Finite Element Method

This finite element method may be introduced as a special case of the Galerkin method for generating algebraic equations. For clarity we consider the discretisation of an elliptic problem at this stage. The original PDE is multiplied by a test function and then integrated to produce a ‘weak’ approximation to the continuous solution. The process of discretising the continuous equation then requires finite dimensional spaces to be defined for both the approximate solution (the trial space) and for the test functions (the test space). For the Galerkin method the set of ‘test’ (or ‘basis’) functions are defined to be equal to the set

of trial functions. For the finite element method these functions must have local support: in the simplest case one basis function is defined for each point in the grid. The specific choice of test and trial spaces for the Galerkin FE method are described below using the example of Poisson's equation,

$$-\overrightarrow{\nabla}^2 u = f,$$

on the domain $\Omega = (0,1) \times (0,1)$, with the boundary condition $u|_{\partial\Omega} = 0$. Here $u(\overrightarrow{x})$ is the unknown function and $f(\overrightarrow{x})$ is known.

The Weak Form

The first step of a FE discretisation is to develop a weighted-integral statement of the governing equation(s). Firstly, the equation is multiplied by some test function, w, say and then integrated:

$$-\int_\Omega (\overrightarrow{\nabla}^2 u) w d\Omega = \int_\Omega f w d\Omega. \tag{2.1}$$

We now wish to eliminate the second derivative in the above expression and can make use of the standard identity

$$\overrightarrow{\nabla} \cdot (w \overrightarrow{\nabla} u) = \overrightarrow{\nabla} w \cdot \overrightarrow{\nabla} u + w \overrightarrow{\nabla}^2 u.$$

We can replace the second order term as follows

$$\int_\Omega w \overrightarrow{\nabla}^2 u d\Omega = \int_\Omega \overrightarrow{\nabla} \cdot (w \overrightarrow{\nabla} u) d\Omega - \int_\Omega \overrightarrow{\nabla} w \cdot \overrightarrow{\nabla} u d\Omega.$$

Using the divergence theorem we can replace the first term on the right-hand side with a boundary integral giving

$$\int_\Omega w \overrightarrow{\nabla}^2 u d\Omega = \int_{\partial\Omega} \overrightarrow{\hat{n}} . (w \overrightarrow{\nabla} u) ds - \int_\Omega \overrightarrow{\nabla} w \cdot \overrightarrow{\nabla} u d\Omega,$$

where $\overrightarrow{\hat{n}}$ is the outward unit normal. Since we have Dirichlet boundary conditions throughout $\partial\Omega$ it turns out to be necessary to restrict the test functions, w, to be zero on the boundary, making the boundary integral zero. Substituting in the new expression for the left-hand side of Equation (2.1) gives:

$$\int_\Omega \overrightarrow{\nabla} w \cdot \overrightarrow{\nabla} u d\Omega = \int_\Omega f w d\Omega, \tag{2.2}$$

for any w in the test space. This weighted-integral statement along with the specified boundary conditions of the problem constitute the weak form of the governing equations [76]. The weak form is so called because, the weak solution, u, can be less differentiable than the original.

Spatial Discretisation

In order to discretise the weak form we approximate the unknown solution u with a function, $\overline{u}$, from a finite dimensional trial space. For this particular version of the method the finite dimensional space must be continuous (although the first derivative may be discontinuous). Given such a trial space and a basis for it ($N_1, ..., N_n$ say) then we may write:

$$\overline{u} = \sum_{i=1}^{n} u_i N_i,$$

where u_i is to be determined for $i = 1, ..., n$ and n is the size of the system. Now, the Galerkin method is defined by choosing the test functions in Equation (2.2) to be the same basis functions used to define the trial space. Hence the discretised equation is:

$$\int_\Omega \overrightarrow{\nabla} \overline{u} \cdot \overrightarrow{\nabla} N_j d\Omega = \int_\Omega f N_j d\Omega,$$

for $j = 1, \ldots, n$. Using the definition of the $\overline{u}$ we can take the values u_i out of the integral giving:

$$\sum_{i=1}^{n} u_i \int_\Omega \overrightarrow{\nabla} N_i \cdot \overrightarrow{\nabla} N_j d\Omega = \int_\Omega f N_j d\Omega. \tag{2.3}$$

Finite Element Basis Functions

Various types of basis functions can be employed in the Galerkin method depending on the dimension of the problem. The key feature of a finite element method is that the basis functions should only have a very restricted support so that the vast majority of integrals $\int_\Omega \overrightarrow{\nabla} N_i \cdot \overrightarrow{\nabla} N_j d\Omega$ are zero where $i \neq j$. This is achieved through breaking down the problem domain into a set of subdomains called elements. A given basis function N_i is associated with a point i in the computational grid where it takes a value of 1. This basis function will take a value of 0 at all other points in the grid. The definition of the basis function is:

$$N_j(x_i) = \begin{cases} 1 & if (i = j) \\ 0 & if (i \neq j) \end{cases} \tag{2.4}$$

where x_i is the position vector of point i of a finite element grid. Therefore the function is only non-zero over the elements that have the grid point j as a vertex. The span of the set of n basis functions forms the FE trial space. The simplest example of a finite element trial space comes from the use of linear basis functions, which produce a piecewise linear approximation to the solution. It is also possible to produce basis functions of higher order for a more sophisticated approximation. For example they could be constructed in such a way so as to make the approximating function quadratic or cubic on each element. We can also apply the basis function definition (2.4) in any number of spatial dimensions. in particular these piecewise linear basis functions can be used with triangular elements in two-dimensions and with tetrahedral elements in three dimensions. In this thesis we restrict attention to problems on two-dimensional domains broken down into quadrilateral elements. For simplicity we will use bilinear basis functions. These are defined to be the

Cartesian product of a one-dimensional linear basis function in x and a one-dimensional linear basis function in y.

Algebraic System Assembly

Having selected a set of basis functions it is straightforward to convert Equation (2.3), into a matrix form that clearly expresses the system of linear equations. The system may be expressed as:

$$A\underline{u} = \underline{b} \tag{2.5}$$

where $A_{ij} = \int_\Omega \overrightarrow{\nabla} N_j \cdot \overrightarrow{\nabla} N_i d\Omega$ is known as the FE stiffness matrix, $b_j = \int_\Omega f N_j d\Omega$, and $\underline{u}$ is the vector of nodal solution values. In a standard finite element assembly algorithm a loop would be taken over the elements to compile the contributions to b for each j, and to A for each combination of i and j. After this loop over the elements, each internal grid point j has received a contribution to its b_j value for each of the surrounding elements and also a contribution to each of the matrix entries A_{ij}, where i only corresponds to points at the vertices of the surrounding elements.

Finite Element Discretisation of Nonlinear Elliptic PDEs

For a nonlinear elliptic PDE, such as

$$\overrightarrow{\nabla} \cdot (u^2 \overrightarrow{\nabla} u) = f$$

for example, we follow the same approach as for the linear elliptic problem above. This will produce the discrete equations

$$\int_\Omega \overline{u}^2 (\overrightarrow{\nabla} N_j \cdot \overrightarrow{\nabla} \overline{u}) d\Omega = \int_\Omega f N_j d\Omega,$$

for $j = 1,...,n$. In this case the vector of discrete solution values cannot be extracted from the left-hand side integral in such a straightforward manner. Systems of nonlinear equations, such as this, will be represented throughout this thesis as:

$$\underline{K}(\underline{u}) = \underline{b}, \tag{2.6}$$

where $\underline{K}(\underline{u}) = [k_1(\underline{u}),...,k_n(\underline{u})]^T$, and $k_j(\underline{u}) = \int_\Omega \overline{u}^2(\overrightarrow{\nabla} N_j \cdot \overrightarrow{\nabla}\overline{u})d\Omega$.

Finite Element Discretisation of Time-Dependent PDEs

An example of a simple time-dependent problem is:

$$\frac{\partial u}{\partial t} = \overrightarrow{\nabla}^2 u,$$

for $(x,y,t) \in (0,1)\times(0,1)\times(0,T] = \Omega\times(0,1]$, and subject to boundary conditions $u|_{\partial\Omega} = 0$ and an initial condition $u(x,y,0) = u_0(x,y)$. In this case the finite element discretisation of the problem will produce the system of ODEs:

$$\int_\Omega \frac{\partial \overline{u}}{\partial t} N_j d_\Omega = \int_\Omega \overrightarrow{\nabla} N_j \cdot \overrightarrow{\nabla}\overline{u} d\Omega$$

for $j = 1,...,n$. In this case we can construct the following matrix form:

$$M\underline{\dot{u}} = L\underline{u}$$

where $M_{ij} = \int_\Omega N_i N_j d\Omega$, $L_{ij} = \int_\Omega \overrightarrow{\nabla} N_j \cdot \overrightarrow{\nabla} N_i d\Omega$, $\underline{u}$ is vector of the nodal solution values and $\underline{\dot{u}}$ is the vector of temporal derivatives of u at the n grid points. The discretisation of the temporal derivatives is discussed in Section 2.2.2.

2.2.2 Temporal Discretisation

A large part of this work is focused on two-dimensional parabolic problems which, once the spatial discretisation is complete, produce initial value problems with a continuous temporal variable (i.e. systems of ordinary differential equations (ODEs)). We therefore need to investigate appropriate methods for discretising in time. The two main classes of temporal discretisation techniques are explicit methods and implicit methods. Explicit methods (for example explicit Runge-Kutta, explicit multistep, etc, [22]) are conceptually straight forward and relatively simple to implement. Unfortunately where explicit methods are used the stability of the solution is dependent on the size of the step taken in time (i.e they are only conditionally stable). In particular, the finer the mesh size (Δx) in the spatial discretisation the smaller the time step (Δt) must be (typically for parabolic problems $\Delta t \propto (\Delta x)^2$). This clearly creates a significant amount of computational work in cases where fine grid spacing is required since extremely small time steps are needed. Implicit methods do not have this restriction since those used in practice are unconditionally stable. However, they are typically more complicated to implement. In the following two sections a small selection of some of the most popular explicit and implicit methods will be illustrated using the general initial value ODE system,

$$\dot{\underline{u}} = \underline{f}(t, \underline{u}), \tag{2.7}$$

where $\dot{\underline{u}}$ denotes the derivative in time.

Explicit Methods

The most straightforward example of an explicit technique is the forward Euler method where, given Equation (2.7), we define:

$$\underline{u}^{k+1} = \underline{u}^k + \Delta t\, \underline{f}(t^k, \underline{u}^k), \tag{2.8}$$

where $\underline{u}^k$ is the solution at the old (or initial) time step, $\underline{u}^{k+1}$ is the unknown new solution and Δt is the size of the time step. The forward Euler scheme can be derived simply from the Taylor series expansion:

$$\underline{u}(t^k + \Delta t) = \underline{u}(t^k) + \Delta t \dot{\underline{u}}(t^k) + \mathscr{O}(\Delta t^2).$$

If we then use Equation (2.7) to substitute $\underline{f}(t^k, \underline{u}^k)$ for $\dot{\underline{u}}(t^k)$ and recognise that $\underline{u}(t^k + \Delta t) = \underline{u}(t^{k+1}) = \underline{u}^{k+1}$ then we reach Equation (2.8). The local truncation error for the forward Euler method is $\mathscr{O}(\Delta t^2)$ making it a first order scheme globally. This method is described in more detail in Burden and Faires' book [22]. Since most of the values in the system produced by this method are taken from the old time step this is a straightforward and fast method for taking steps through time.

The Runge-Kutta (R-K) methods [22] provide explicit time stepping methods of higher orders. The midpoint rule, of the form:

$$\underline{u}^{k+1} = \underline{u}^k + \underline{f}\Delta t(t^k + \frac{\Delta t}{2}, \underline{u}^k + \frac{\Delta t}{2}\underline{f}(t^k, \underline{u}^k)),$$

is an example of a second order explicit R-K method. A fourth order explicit R-K approximation is given by:

$$\underline{q}_1 = \Delta t \underline{f}(t^k, \underline{u}^k),$$

$$\underline{q}_2 = \Delta t \underline{f}(t^k + \frac{\Delta t}{2}, \underline{u}^k + \frac{1}{2}\underline{q}_1),$$

$$\underline{q}_3 = \Delta t \underline{f}(t^k + \frac{\Delta t}{2}, \underline{u}^k + \frac{1}{2}\underline{q}_2),$$

$$\underline{q}_4 = \Delta t \underline{f}(t^k + \Delta t, \underline{u}^k + \underline{q}_3),$$

$$\underline{u}^{k+1} = \underline{u}^k + \frac{1}{6}(\underline{q}_1 + 2\underline{q}_2 + 2\underline{q}_3 + \underline{q}_4).$$

Clearly the additional accuracy of this method must be paid for through added computational cost, however when the solution is sufficiently smooth (and stability is not an issue) this scheme is generally superior to its lower order counterparts.

Implicit Methods

The backward Euler method is a first order implicit scheme and as such is the simplest available. Using equation (2.7), the Backward Euler method can be represented as:

$$\underline{u}^{k+1} = \underline{u}^k + \Delta t\, \underline{f}(t^{k+1}, \underline{u}^{k+1}). \tag{2.9}$$

In contrast to the explicit methods most of the values in this system are taken from the new, unknown, time step. This makes the solution of the system much more complicated. If f is a nonlinear function of its second variable the resulting algebraic system will also be nonlinear. In the explicit case this nonlinearity would be suppressed but for the implicit methods to be effective we must find efficient ways to solve the resulting nonlinear algebraic systems.

There are also many higher order implicit methods. The Trapezoidal and Gauss rules are both examples of second order implicit Runge-Kutta schemes. The Trapezoidal rule, with respect to Equation (2.7), is

$$\underline{u}^{k+1} = \underline{u}^k + \frac{\Delta t}{2}\underline{f}(t^k, \underline{u}^k) + \frac{\Delta t}{2}\underline{f}(t^{k+1}, \underline{u}^{k+1}). \tag{2.10}$$

There is a small price to be paid for the higher accuracy of approximation, since we must calculate f for the old time values as well as solving for the new time values implicitly. One more level of complication can be found in the Gauss rule,

$$\underline{u}^{k+1} = \underline{u}^k + \frac{\Delta t}{2}\underline{f}(\frac{1}{2}(t^k + t^{k+1}), \frac{1}{2}(\underline{u}^k + \underline{u}^{k+1})),$$

where the values put into f are linear interpolants of the values at old and new time steps. This adds computational complexity as both sets of values must be maintained throughout the solution process. In many cases however, the second order methods pay for themselves in increased accuracy and a reduction in numerical diffusion.

In order to make a compromise between explicit and implicit methods we can employ the theta method. Applying the theta method to Equation (2.7) gives the form

$$\underline{u}^{k+1} = \underline{u}^k + \theta \Delta t \, \underline{f}(t^{k+1}, \underline{u}^{k+1}) + (1-\theta)\Delta t \, \underline{f}(t^k, \underline{u}^k)$$

where $0 \leq \theta \leq 1$. When $\theta = 0$ this expression simplifies to the forward Euler method (given in Equation (2.8)), when $\theta = 1$ it is the backward Euler method (given in Equation (2.9)) and when $\theta = \frac{1}{2}$ it is the trapezoidal rule (given in Equation (2.10)). When $\theta < \frac{1}{2}$ but $\theta \neq 0$ the stability of the scheme is dependent on the size of the time step and there is an extra expense incurred by calculating the $\underline{f}(t^{k+1}, \underline{x}^{k+1})$ term at each step. This range of θ values therefore have limited use. However, in the range $\frac{1}{2} \leq \theta < 1$ the scheme is unconditionally stable and uses both $\underline{f}(t^k, \underline{x}^k)$ and $\underline{f}(t^{k+1}, \underline{x}^{k+1})$, although it is only second order when $\theta = \frac{1}{2}$. It is possible to vary the value of theta from one time step to the next [17].

Despite the relative complexity of implicit methods they are often used. The absence of a time-step size restriction allows faster simulations in cases where the grid refinement must be very small to achieve the necessary accuracy in space.

Time Discretisation in This Work

In the parabolic examples encountered in this work we always use the approach described above whereby the spatial operators are discretised with the FE method and the resulting

initial value problem is solved using one of the schemes described (or variations thereof). A different approach would be to extend the finite element method used for the spatial variables into another dimension (time) [24, 30]. This is a pleasant, unified method but one which typically requires a larger amount of computational effort. Hence it is not pursued further here.

2.3 Solution of Algebraic Systems

The spatial discretisation techniques presented in Section 2.2.1, when applied to elliptic problems, produce a system of algebraic equations which must then be solved. In the case of parabolic equations there is an ODE system to be solved and, as mentioned in Section 2.2.2, time discretisation produces algebraic systems at each time step. This section will look at some of the available tools for the solution of both linear and nonlinear systems of algebraic equations. The straightforward direct methods for linear systems are illustrated in Section 2.3.1 followed by the more flexible iterative methods for both linear and nonlinear systems in Section 2.3.2. In the following Section, 2.4, multigrid methods are introduced and described in some detail.

2.3.1 Direct Methods

The simplest methods for solving linear algebraic systems are direct methods. These are algorithms with a fixed number of operations (dependent upon the system size, n) which essentially invert the operator matrix to produce a solution. A classic example of a direct method is LU factorisation. Here the matrix, A of a linear system, such as that appearing in Equation (2.5), is broken down into lower and upper triangular component matrices: $A = LU$, by Gaussian elimination [40, pp 92-104]. Two systems are then solved,

comprising of the upper and lower matrices, i.e.

$$L\underline{z} = \underline{b} \tag{2.11}$$

$$U\underline{u} = \underline{z} \tag{2.12}$$

Firstly Equation (2.11) is solved by forward substitution for the vector $\underline{z}$. This vector is then used in the solution of Equation (2.12), by backwards substitution, for the final solution vector $\underline{u}$. Since L and U are both triangular matrices the cost of each of these solves is only $\mathcal{O}(n)$. Again, more details can be found in [22].

The drawback of Direct methods is the amount of computational work and computer memory required. If n is the number of unknowns in the system, a direct method theoretically requires $\mathcal{O}(n^3)$ operations to find a solution (for example when LU factorisation is used the elimination step requires $\mathcal{O}(n^3)$ operations). However for the numerical solution of PDE's, using finite difference or finite element discretisation, the sparse matrices produced will only require $\mathcal{O}(n^2)$ operations. Therefore, as n increases, direct methods eventually become intractable.

Due to the local nature of the FE basis functions (described in Section 2.2.1) the matrices arising from these methods are sparse. The iterative methods, which are presented in the next section, are a more attractive option than direct methods, for these sparse systems: although some direct methods may be modified to be suitable for sparse matrices [58].

2.3.2 Iterative Methods

Iterative methods are a desirable alternative to direct methods for the solution of sparse systems of equations. The iterative approach taken to the solution of algebraic systems can be summarised as follows. An initial guess of the solution is taken and then some

form of correction is calculated and used to improve this guess. This is done repeatedly until the accuracy of approximation (typically measured in terms of the residual which is a quantity that may always be calculated) meets some criteria. Iterative solution methods can be divided into those designed for linear algebraic systems and those which will be effective on nonlinear systems. Examples of both types of solver are given below.

Iterative Methods for Linear Systems

A few examples of iterative solvers for linear systems will be illustrated here, using the linear system in Equation (2.5). Two of the most established iterative methods are the Jacobi and Gauss-Seidel (G-S) methods. The Jacobi iteration [22] can be given in the following matrix form. Taking the linear system in Equation (2.5) we can split up the matrix A as follows:

$$A = D - L - U,$$

where D is the diagonal of A and L and U are the negated, strictly lower and upper triangular sections of A respectively. This gives the linear system:

$$(D - L - U)\underline{u} = \underline{b},$$

which can be rearranged to:

$$\underline{u} = D^{-1}(L + U)\underline{u} + D^{-1}\underline{b},$$

(provided that all diagonal entries of A are non-zero). If $\underline{u}^{(r)}$ is the approximate solution at iteration r and $\underline{u}^{(r+1)}$ is the improved approximation at iteration $r+1$, we can characterise one iteration of the Jacobi method as:

$$\underline{u}^{(r+1)} = D^{-1}(L + U)\underline{u}^{(r)} + D^{-1}\underline{b}. \tag{2.13}$$

The G-S iterative technique is a modified version of the Jacobi method which takes advantage of the elements of $\underline{u}^{(r+1)}$ as they are calculated. The component form of this iteration is:

$$u_j^{(r+1)} = \frac{1}{A_{jj}}\left[-\sum_{i=1}^{j-1}(A_{ij}u_i^{(r+1)}) - \sum_{i=j+1}^{n}(A_{ij}u_i^{(r)}) + b_j \right]. \tag{2.14}$$

Since the improved approximations are used at an earlier stage, the G-S iteration generally converges faster than the Jacobi version.

The Successive Overrelaxation (SOR) method came from an acceleration of the G-S method [22]. The new approximation, in this case, is a weighted average between the previous approximation and the G-S correction. The iteration can be characterised by the expression,

$$\mathfrak{u}^{(r+1)} = \omega\overline{\mathfrak{u}}^{(r)} + (1-\omega)\mathfrak{u}^{(r)},$$

where $\overline{\mathfrak{u}}$ is the G-S correction.

The set of Krylov subspace methods were a later development, including the Conjugate Gradient (CG) method [6,40] for symmetric positive definite systems and the Generalized Minimal Residual (GMRES) method for non-symmetric systems [83]. The CG method minimises a functional $f(\underline{u}) = \frac{1}{2}(\underline{r}, A^{-1}\underline{r})$, where $\underline{r}$ is the residual vector: $\underline{r} = \underline{b} - A\underline{u}$ and $(\cdot,\cdot)$ is an inner product. Minimising this functional is equivalent to solving the system. The solution is improved iteratively. At each iteration a search direction, $\underline{d}^{(r)}$ is chosen, along with a scalar $\tau^{(r)}$ designed to optimise the movement of the solution along that search direction. The solution is then improved by $\underline{x}^{(r+1)} = \underline{x}^{(r)} + \tau^{(r)}\underline{d}^{(r)}$. Each iteration of the solver requires a new search direction. The first direction is chosen to be the residual vector and each of the following directions is chosen to be orthogonal to all of the previous ones through an efficient two-term recurrence relation. The iterations of this solver are therefore equivalent to the steps of building up the Krylov space associated with

A and $\underline{r}^0$. CG is now the method of choice for symmetric, positive definite systems.

For a comprehensive survey of the development of iterative methods for linear systems, see [84].

Iterative Methods for Nonlinear Systems

Newton's method is the best known technique for solving systems of nonlinear equations. This method is illustrated using a nonlinear system with all terms on the left-hand side: $\underline{K}(\underline{u}) = \underline{0}$, where $\underline{K}(\underline{u}) = [k_1(\underline{u}), ..., k_n(\underline{u})]^T$ and n is the size of the system. Taking $\underline{s} = \underline{u} - \underline{v}$ where $\underline{v}$ is the approximate solution and $\underline{u}$ is the exact solution, makes $\underline{s}$ a correction to the approximation. We can derive Newton's method using a Taylor series expansion of the nonlinear system $\underline{K}(\underline{v} + \underline{s}) = \underline{0}$ giving:

$$\underline{K}(\underline{v} + \underline{s}) \approx \underline{K}(\underline{v}) + J(\underline{v})\underline{s},$$

where the terms higher than first order have been ignored and $J(\underline{v})$ is the Jacobian matrix defined as:

$$J(\underline{v}) = \begin{bmatrix} \frac{\partial k_1}{\partial v_1} & \frac{\partial k_1}{\partial v_2} & .. & \frac{\partial k_1}{\partial v_n} \\ \frac{\partial k_2}{\partial v_1} & \frac{\partial k_2}{\partial v_2} & .. & \frac{\partial k_2}{\partial v_n} \\ : & : & & : \\ \frac{\partial k_n}{\partial v_1} & \frac{\partial k_n}{\partial v_2} & .. & \frac{\partial k_n}{\partial v_n} \end{bmatrix}.$$

Solving the system,

$$J(\underline{v})\underline{s} = -\underline{K}(\underline{v}), \tag{2.15}$$

allows us to form a first order correction to the approximate solution. A comprehensive description of Newton's method can be found in [72] along with a variation on this known as the Secant method.

Newton's method and the Krylov methods can be combined in the hybrid Newton-Krylov methods [54] for nonlinear systems. Newton's method is used to linearise the system as described above and the Krylov iteration is used as an inner iteration to solve Equation (2.15).

2.4 Multigrid

Multigrid methods use various 'levels' of the computational grid to speed up the convergence of an iterative solver for systems of equations derived from a finite difference, element or volume discretisation of a PDE. The error in any system is composed of various frequencies. For a particular class of iterative solvers, with the so-called 'smoothing' property the high frequency components are eliminated first and relatively quickly. The majority of iterations are then taken in eliminating the remaining low frequency error components. Multigrid methods use a hierarchy of increasingly coarse grids, as illustrated in Figure 2.1, to reduce the work done in eliminating the lower frequency components of the error. Once the high frequency components have been removed, on a given grid, the problem is defined on a coarser grid in the hierarchy. The highest frequencies on this coarser grid correspond to the non-damped error components on the finer grid and these are eliminated in the first few coarse grid iterations. The problem is then passed to a coarser grid still. After the contribution to the solution is found on the coarsest grid it is passed back to the finer grids in turn. Using the coarser grids in this way produces an accurate solution in less fine-grid iterations. Since much of the work is being performed on coarser grids, it is also relatively inexpensive. The cost of a well posed multigrid method is proportional to the number of unknowns in the finest grid. Furthermore, the convergence rate of a multigrid solver is independent of the size of the system over the finest grid. Hence the method has a total cost that grows only linearly with the number of unknowns in the system.

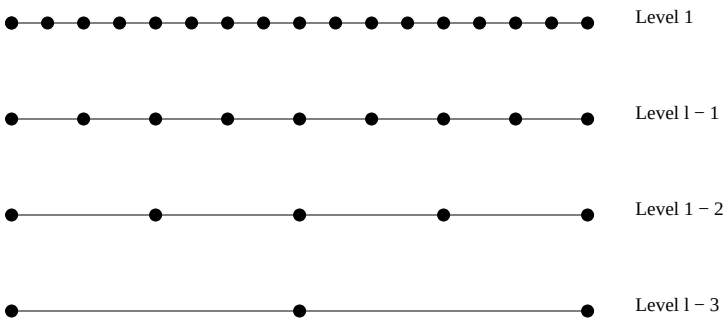


Figure 2.1: A hierarchy of non-uniform grids ranging from the finest at level l to the coarsest at level $l-3$.

Before describing some particular multigrid methods we must clarify the nature of the hierarchy of grids. For the purposes of this section it is assumed that every grid in the hierarchy is uniform. (The use of non-uniform grids is discussed in Section 2.6.) In the uniform case, a grid level, ℓ, is taken to be the grid with spacing $\Delta x = \frac{dom}{2^\ell}$, where dom is the size of the computational domain. All of the grid hierarchies used throughout this work are composed of 'nested' grids, i.e. the set of points in any one grid is a subset of those in any of the finer grids.

The tutorial [19] by Briggs *et al* gives a comprehensive introduction to multigrid methods. Trottenburg [92] also includes material on multigrid applied to more complex problems. There are two main types of multigrid method, the Defect Correction method and the Full Approximation Scheme (FAS). These are intended for linear and nonlinear problems respectively. The Defect Correction method can be used to solve nonlinear problems by using it as an 'inner iteration' of a linearising method [63] such as Newton's method (described in Section 2.3.2). However, FAS multigrid has the advantage of being applied directly to the nonlinear problem in a 'matrix free' manner. It has been successfully applied to droplet spreading problems in [38], aerodynamic design [48] and plasticity problems [33], among many others. Details of both of these methods are presented below.

2.4.1 The Defect Correction Method

As the defect correction method is designed for linear systems we will use the generic system of Equation (2.5) to illustrate the method. This system of equations is assumed to be derived from the discretisation of an elliptic PDE on the finest of a series of uniform nested grids as in Figure 2.1. An easy way to assess the solution error is to look at the residual (or defect) of the system. The form of the residual is $\underline{r} = \underline{b} - A\underline{v}$ and, when $\underline{v}$ is an approximation to $\underline{u}$, the residual $\underline{r}$ will be non-zero. We can say that $\underline{u} - \underline{v} = \underline{e}$ where $\underline{e}$ is the error in the approximation. We can therefore form an equation in terms of the residual and the error, called the residual equation:

$$A\underline{e} = \underline{r}. \tag{2.16}$$

This equation can be solved to produce an error with which to correct the approximated solution of the original system. Once an initial approximate solution has been found on the fine grid the defect correction method uses this to form the residual equation (2.16), on the coarse grid. The residual equation is then solved for the error e. The interpolant of this error onto the fine grid is used to correct the solution of the original system. Algorithm 1 illustrates one iteration of a two-grid defect correction scheme. The following sections describe some of the key aspects of the algorithm.

Algorithm 1 Two-Grid Defect Correction Method

1: ℓ = level number of current grid
2: $L = \ell - 1$ (level number of grid one coarser than ℓ)
3: Choose initial fine grid approximation $\underline{v}^\ell$
4: Calculate $\underline{b}^\ell$
5: Update $\underline{v}^\ell$ by smoothing ρ_1 times with the system $A^\ell \underline{u}^\ell = \underline{b}^\ell$
6: Find residual $\underline{r}^\ell := \underline{f}^\ell - A^\ell \underline{v}^\ell$
7: Restrict residual to create coarse right-hand side $\underline{b}^L := \tilde{I}_\ell^L(\underline{r}^\ell)$
8: Solve the system $A^L \underline{e}^L = \underline{b}^L$ with the initial guess of $\underline{e} = \underline{0}$
9: Prolongate the error calculated $\underline{e}^\ell := I_L^\ell(\underline{e}^L)$
10: Correct the fine grid approximation $\underline{v}^\ell = \underline{v}^\ell + \underline{e}^\ell$
11: Update $\underline{v}^\ell$ by smoothing ρ_2 times with the system $A^\ell \underline{u}^\ell = \underline{b}^\ell$

Pre- and Post-Smoothing

The pre- and post-smooths are sweeps of the iterative solver taken before and after the coarse grid correction, at lines 5 and 11 respectively of Algorithm 1. The pre-smoothing produces an initial, fine-grid, approximation to the solution. The intention is that the high frequency error has been eliminated from this approximation using fine grid smooths. The post-smoothing is simply to re-adjust the solution after the coarse grid correction (removing any high frequency components that the correction may have introduced). The number of pre- and post-smooths, ρ_1 and ρ_2 respectively, depends on the application but will typically range from zero to three.

Inter-grid Transfer Functions

Inter-grid transfer functions are the methods by which data is passed from one grid level to another. We need to define two types of inter-grid transfer: a 'restriction' function I_ℓ^L which will transfer the residual from the fine grid to the coarse grid and a 'prolongation' function I_L^ℓ which will transfer the correction from the coarse grid to the fine grid. The simplest approach to restriction is injection, illustrated in Figure 2.2, where values are transferred directly. A standard alternative is Full Weighting, where a weighted average of surrounding nodes is taken. This can be seen in Figure 2.3. Our method of prolongation, which employs linear interpolation to find the values of points that do not exist on the coarse mesh, is shown in Figure 2.4. There is an extra complication when using FE methods as the residual is an element-wise quantity dependent on the element support area. Consequently, as well as being restricted the residual must also be scaled to match the relative size of the element support area on the coarse grid. (The tilde above the operator used to restrict the residual, $\tilde{I}_\ell^L$, indicates that this scaling is also performed.)

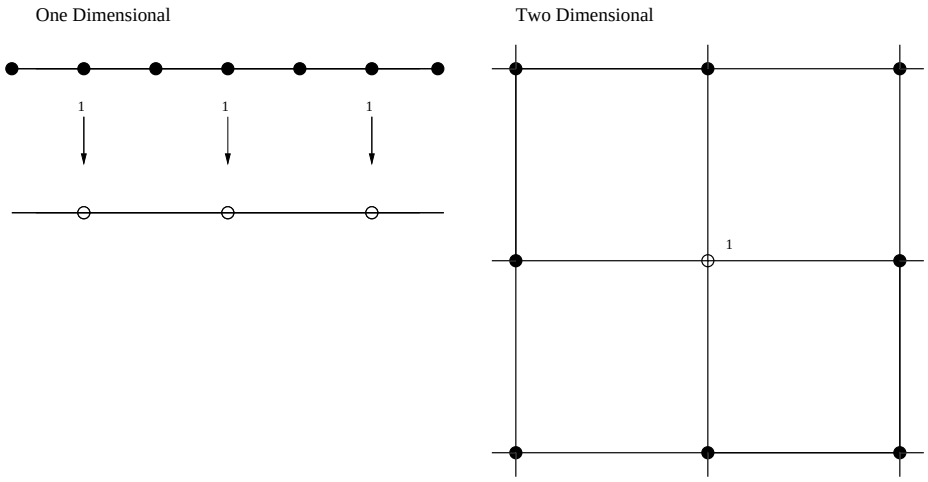


Figure 2.2: Multigrid restriction by injection: in this case the values of points on the coarse grid take the value of the corresponding points on the fine grid. Any values held at fine grid points that do not correspond to coarse grid points are ignored.

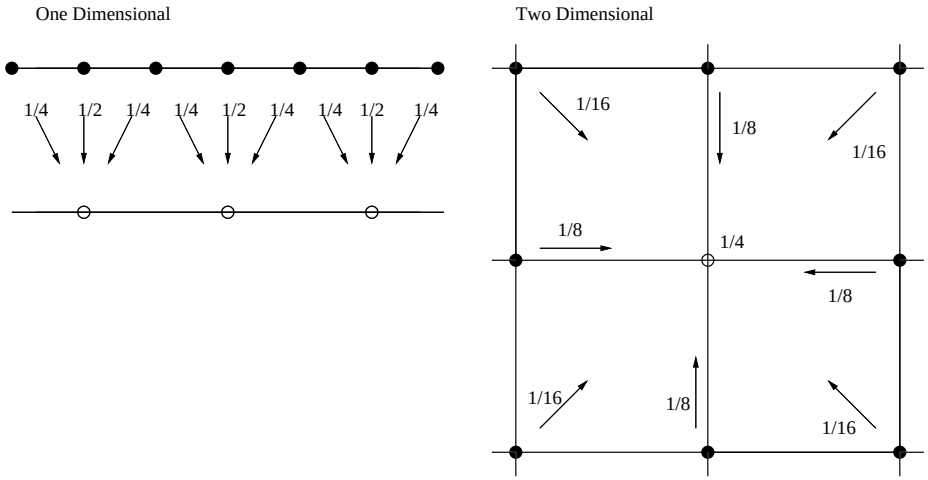


Figure 2.3: Multigrid restriction using full weighting: Where a fine grid point corresponds to a coarse grid point, its value, along with those of its fine grid neighbours, are used (with various weightings) to produce a value for the corresponding coarse grid point. The weightings, shown here for the one-dimensional and two-dimensional cases, always sum to 1.

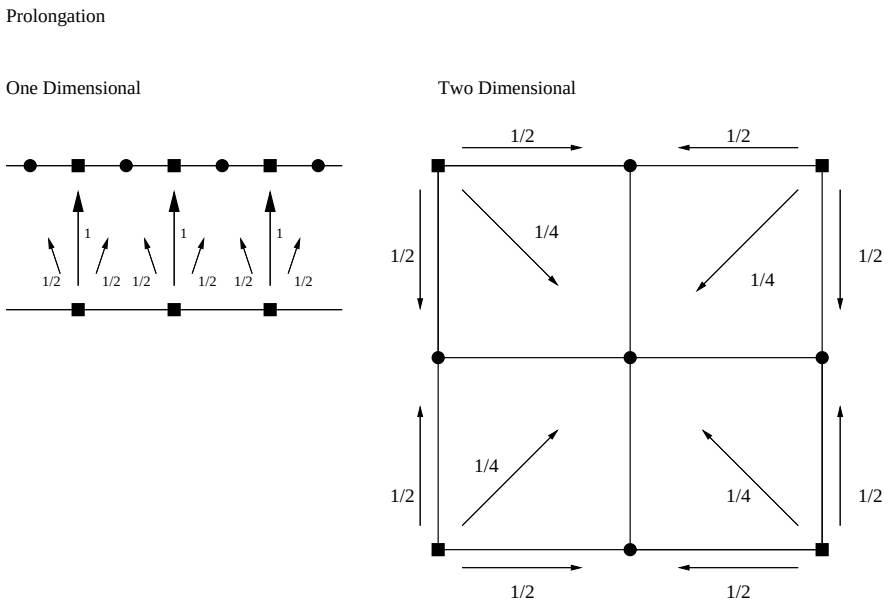


Figure 2.4: Multigrid prolongation: Fine grid points with corresponding coarse grid points take their value directly from this corresponding point. Fine grid point with no coarse grid counterpart are found by linear interpolation.

Extension to Full V-Cycle

We can extend the two-level algorithm in Algorithm 1 so that a hierarchy of grids is used recursively. In this case each grid is pre-smoothed and the residual is restricted to the next, coarser, grid until the coarsest grid is reached. On the coarsest grid the problem is solved exactly. Then each grid receives a correction, performs post-smooths and passes the improved correction to the next, finer, grid. The multigrid algorithm is normally implemented so that there can be many iterations of this recursive 'V-Cycle'. However, only a small number are generally required. The V-Cycle is one of several possible cycle types which can be employed within the multigrid method. Another example is the W-Cycle which uses the same inter-grid transfer operation for the solution and residual but orders the transfers differently.

2.4.2 The Full Approximation Scheme

The Full Approximation Scheme (FAS) is so called as the complete original problem is solved on every level of the grid throughout the multigrid cycle (rather than just a defect equation). This comes from the need to solve nonlinear problems. A generic nonlinear system will be represented as $\underline{K}(\underline{u}) = \underline{b}$. The operator K is nonlinear, the right-hand side vector $\underline{b}$ is known and $\underline{u}$ is the unknown solution. Since the operator is not linear we cannot assume that it will be distributive over the vectors. Although we know that $\underline{u} - \underline{v} = \underline{e}$, we cannot therefore conclude that $\underline{K}(\underline{u}) - \underline{K}(\underline{v}) = \underline{K}(\underline{e})$. Taking the usual form of the residual $\underline{r} = \underline{b} - K(\underline{v})$ and substituting using the original system gives a new residual equation,

$$\underline{K}(\underline{u}) = \underline{r} + \underline{K}(\underline{v}), \tag{2.17}$$

where $\underline{r}$ is the residual and $K(\underline{v})$ is the operator applied to the current approximation, $\underline{v}$. Using these components we can solve for $\underline{u}$. The solution vector u can be thought of as $(\underline{v} + \underline{e})$, a corrected version of the approximation.

The multigrid FAS solves equation (2.17), which is effectively an amended version of the original system, on the coarse grid. This produces an improved approximation to the solution. A correction can then be calculated on the coarse grid and passed to the fine grid. Algorithm 2 is a two-grid version of the FAS scheme. The components of this algorithm are much the same as the defect correction algorithm in Section 2.4.1. In fact, when applied to a linear system corresponding to the discretisation of a linear PDE the FAS scheme is equivalent to defect correction.

In order to form the right-hand side of Equation (2.17) on the coarse grid we must restrict both the residual ($\underline{r}^{\ell}$) and the approximate solution ($\underline{v}^{\ell}$). As mentioned in the previous section $\underline{r}^{\ell}$ is restricted with the operator $\tilde{I}_{\ell}^{L}$ which scales values to resolve the difference

in element support sizes between the grids. As for the defect correction method this two-grid algorithm can be extended to make use of a hierarchy of grids in an order such as the V-Cycle.

Algorithm 2 Two-Grid Full Approximation Scheme

1: ℓ = level number of current grid
2: $L = \ell - 1$ (level number of grid one coarser than ℓ)
3: Choose initial fine grid solution $\underline{v}^{\ell}$
4: Calculate the right-hand side $\underline{b}^{\ell}$
5: Update $\underline{v}^{\ell}$ by smoothing ρ_1 times with the system $K^{\ell}(\underline{u}^{\ell}) = \underline{b}^{\ell}$
6: Find residual $\underline{r}^{\ell} := \underline{b}^{\ell} - K^{l}(\underline{v}^{\ell})$
7: Restrict residual $\underline{r}^{L} := \tilde{I}_{\ell}^{L}(\underline{r}^{\ell})$
8: Restrict solution $\underline{v}^{L} := I_{\ell}^{L}(\underline{v}^{\ell})$
9: Calculate coarse grid right-hand side $\underline{b}^{L} := \underline{r}^{L} + K^{L}(\underline{v}^{L})$
10: Save the initial coarse grid solution $\underline{v}^{L}_{init} = \underline{v}^{L}$
11: Solve the system $K^{L}(\underline{v}^{L}) = \underline{b}^{L}$
12: Calculate the coarse grid correction $\underline{e}^{L} = \underline{v}^{L} - \underline{v}^{L}_{init}$
13: Prolongate the correction $\underline{e}^{\ell} := I_{L}^{\ell}(\underline{e}^{L})$
14: Correct the fine grid solution $\underline{v}^{\ell} = \underline{v}^{\ell} + \underline{e}^{\ell}$
15: Update $\underline{v}^{\ell}$ by smoothing ρ_2 times with the system $K^{\ell}(\underline{u}^{\ell}) = \underline{b}^{\ell}$

2.5 Adaptivity

The aim of adaptivity is to provide an increased accuracy of approximation in certain areas of the spatial domain or at certain time steps. This work will focus on spatial adaptivity in the solution of elliptic and parabolic equations and systems. Increased accuracy of approximation is required in areas of the spatial domain where there is a shock, singularity or any other sudden change in the solution or its derivatives. The error in these areas is generally much higher than that elsewhere in the domain if a uniform FE or FD approximation is made. The extra complexity involved in adding spatial adaptivity is therefore justified in these cases since it allows a much more accurate solution to be found for a given computational cost.

2.5.1 Adaptive Techniques

There are several types of spatial adaptivity which can be employed during a numerical solution. We can increase the order of the discretisation. This is referred to as a p-refinement method [70, 94]. In a finite element sense p-refinement involves increasing the order of the basis functions used in the approximation [70]. Increasing the order of the basis functions also increases the size of the system (and reduces the sparsity) and likewise the cost of the solution process. Local p-refinement has therefore been developed, for example [25], where only certain elements have a higher order approximation. Several orders of approximation can be used within one solution method to speed up the convergence, such as the 'p-multigrid' method presented in [36].

Alternatively, the positions of existing mesh points can be improved using movement or relocation as in [8, 45, 68]. This method is referred to as r-refinement. In this case a fixed number of computational points are used and the position of each point is optimised with the aim of minimising the global error.

Probably the simplest and the most widely used of the adaptive methods is that of h-refinement [14, 26, 27, 59, 80]. In this case, mesh generation algorithms are developed which create extra grid points in areas of high solution activity (or where the error is estimated to be large [71]). This creates non-uniform grids with small elements over areas of high error and larger ones elsewhere. The method of h-refinement has been combined with r-refinement as in [60] and also with p-refinement as in [70].

An h-refinement scheme for a triangular grid is given by Rivara in [80]. In that work the basic mechanism for refining a triangle is bisection, where a new point is placed at the centre of the triangle's longest edge and a new edge is created, joining this to the opposite existing point. A general triangle is shown in Figure 2.5, divided into two smaller triangles

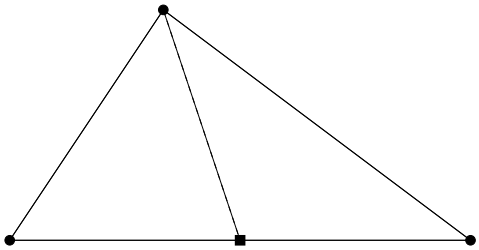


Figure 2.5: Bisection for triangular grids: Creating a new edge joining the mid-point of the longest edge to the opposite vertex.

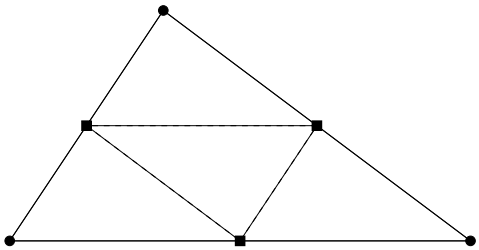


Figure 2.6: Dividing triangular elements into four: Creating a new point at the midpoint of each edge and joining these to divide the element into four smaller elements.

using this bisection technique. This method is applied twice per element in a refinement sweep in order to divide each triangle into four smaller triangles at each refinement. Triangular elements can also be divided into four by joining the midpoints of each of the edges, as illustrated in Figure 2.6. This method is employed by Lohner [59]. Refinement methods which use this hierarchical subdivision of elements produce non-uniform grids where newly created elements are nested inside existing elements. This is not the case for all gridding methods, for example Delaunay Triangulation methods described in [62]. In this case points are placed in the domain according to some error minimising scheme. The mesh is then reformed to connect the new point in a way which maximises the minimum angle of all of the triangles. The Advancing-Front technique, also described in [62], is initiated by meshing the domain boundary (a set of edges in two dimensions or a set of faces in three dimensions). Triangulation is then performed in successive layers into the domain until the whole domain is covered and all points are fully connected.

Adaptively refined meshes are not only created prior to the solution process but often

re-adapted during the solve, especially in the case of time-dependent problems. In this case the solution values (and values at any interpolation points) must be transferred from the old grid to the newly adapted one. Re-meshing techniques such as Delaunay Triangulation would produce a completely different triangulation from the previous grid. This means that transferring the solution values involves searching for the element in the old grid which contains each point in the new grid. Hierarchical meshing schemes (such as [80]) subdivide existing elements meaning that the transfer of solution values is a local operation. A comparison of methods fitting into these two categories is given in [32].

One established family of h-refinement techniques for quadrilateral grids is Adaptive Mesh Refinement (AMR) led by Marsha Berger, amongst numerous others [13, 15, 16]. This technique is presented in [16], using FD methods, for hyperbolic problems. A uniform rectangular initial grid is set up then overlayed with uniform subgrids in areas where the error estimate exceeds a tolerance. These subgrids can, themselves, contain subgrids and can overlap with others of the same level. (Grids of the same level have the same grid spacing.) This grid refinement is achieved by marking points in the initial grid which exceed a desired tolerance in the error estimate and then running a clustering algorithm to put them into groups which will make efficient subgrids. The clustering stage of the process is complex and expensive.

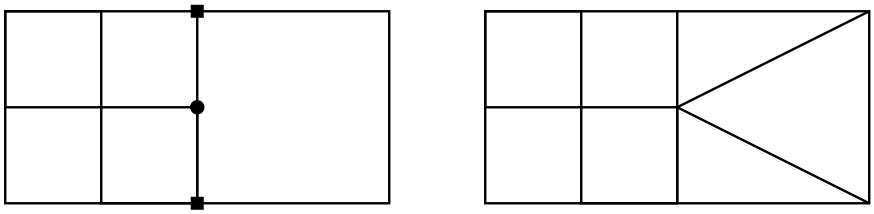
Most of the methods described above can be extended for use in three dimensions. The triangular techniques could be extended for tetrahedral grids [11] and the quadrilateral techniques extended for cubic ones [47]. A three-dimensional version of the AMR technique is used in [13]. and the method of bisection is applied to tetrahedral meshes in [5].

2.5.2 Data Structures for Adaptive Gridding

In the case of r- and h-refinement the resulting grids are non-uniform and often change during the course of the solution. The data structures for these grids must facilitate a search through the points, edges or elements, store the connectivity of these three sets and allow for some analysis of the local area around a point, edge or element. This information can be stored in a set of arrays. For example the edge connectivity in [59] is stored as an array of 'edges' each of which contains the two points which that edge connects. It is possible however to design data structures which hold more grid information inherently in their structure. For example, in [80] Rivara implements a 'molecular list' where each item represents a point and stores an ordered set of neighbouring points as well as whether it is a boundary point. In this case the elements are stored implicitly. (For an FE discretisation, where an assembly over the elements is usually required, it is more appropriate to store the elements explicitly.) Tree data structures are a useful device for storing non-uniform grid information. If levels of the tree are associated with levels of the computational grid then areas of high refinement in the grid can be represented as highly developed branches of the tree. For example, quadtree data structures are used for an implementation of the level set method in [88] and applied to a phase field model in [75]. Each element in the grid is represented as a node in the quadtree and the refinement of an element involves the creation of four child nodes in the tree, each representing a subset of the parent element. More details of these data structures are given in Section 4.2.1.

2.5.3 Hanging Nodes

In a non-uniform grid with a fixed element shape we encounter the problem of disconnected (or 'hanging') nodes, which occur when an element is refined but its neighbour is not. In Figure 2.7 the hanging node is indicated by a filled circle in the grid on the left. Since the hanging node in this example is only connected to two elements and lies on the edge of the neighbouring, larger element, it could cause a discontinuity in a FE solu-



Red Refinement Green Refinement

Figure 2.7: Hanging nodes in quadrilateral grids: Two options for hanging nodes are shown. 'Red' refinement, on the left, leaves these nodes disconnected while 'green' refinement, on the right, connects them using non-standard refinement.

tion along this edge. It is possible to eliminate hanging nodes by adding extra triangular refinement. This is sometimes called 'green' refinement. The right-hand side grid in Figure 2.7 is one example of resolving a hanging node with green refinement. This technique is natural when implementing an unstructured triangular (or tetrahedral) grid. However, resolving hanging nodes in this way within a quadrilateral (or cubic) grid produces grids with more than one element shape (known as 'hybrid' grids). A tree-based quadrilateral gridding system with a triangular, green refinement scheme is presented in [75]. Another example of green refinement can be found in [42]. Green refinement eliminates the problem of hanging nodes but adds complexity to the refinement process. The extra refinement is added once the standard refinement is complete and is generally removed before re-adaption of the grid. As well as complicating the grid generation, this method adds extra work to the solution process. In the cases where hybrid grids are produced the solution process must be able to resolve the various element types in these grids.

An alternative to green refinement is to simply allow certain hanging nodes to exist in the grid (this is sometimes referred to as 'red' refinement). Clearly this method is simpler from a grid generation perspective as no further refinement is required. However, the hanging nodes and the discontinuities that they cause must be dealt with elsewhere in the numerical approximation process. This can be done by modifying the FE basis function

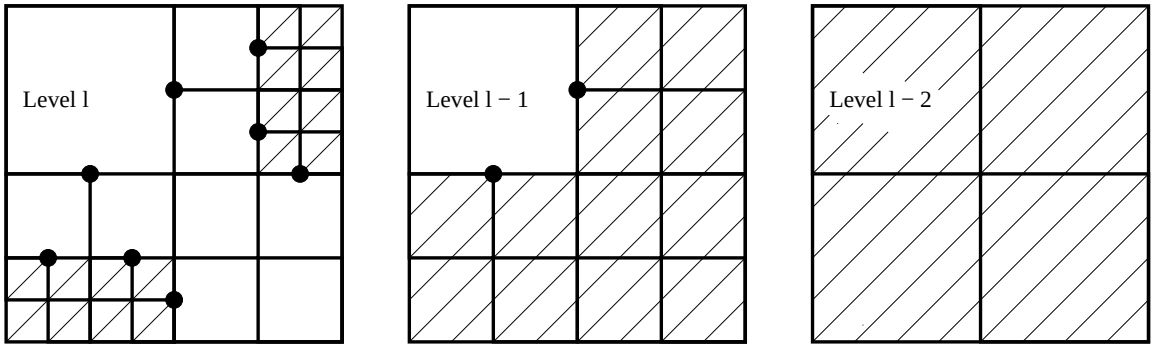


Figure 2.8: Multigrid levels in a non-uniform grid: The shaded areas show the elements which would make up a grid level in the MLAT or FAC definition.

as in [43, 89, 95]. Another possibility is to constrain the values at these hanging nodes, effectively removing their degree of freedom. This technique is used by Rheinboldt and Mesztenyi in [77] (part of the development of the FEARS package). In that work the assembly of the finite element stiffness matrices is enriched with interpolation operators. This method effectively constrains the hanging node values but adds complexity to the assembly of the element matrices. More examples of the restriction of the hanging node values can be found in [47, 67].

2.6 Multigrid Methods for Adaptively Refined Grids

This section presents some of the existing work on the application of multigrid methods to problems defined on adaptively refined grids. Clearly there must be amendments made to the multigrid method in order for it to be effective on the non-uniform grids which result from adaptive refinement. The problems presented by non-uniform grids are reduced to three main points below.

- In a uniform mesh a ‘level’ is defined as containing elements of a particular size. Since elements of several different sizes exist in a single non-uniform mesh, the idea of a ‘level’ is no longer so clearly defined.

- The smoothing technique used must be able to accommodate the inconsistency of

element size and possibly the presence hanging nodes in the grid.

- The inter-grid transfer functions must be specially formulated at points on the boundary between two elements of different sizes.

There have been a variety of approaches to adaptive multigrid. The most established of these is the Multilevel Adaptive Technique (MLAT) of Brandt [18]. In the MLAT scheme a level in the mesh is composed of all of the elements of a particular size and no others. For example, the shaded areas of Figure 2.8. As a result the finer grids may be composed of several 'patches' covering subsets of the domain. When smoothing over level l, temporary Dirichlet boundary conditions are set along the interface between level l elements and level $l-1$ elements. The smoothing is then performed on the uniform subdomains only. Using this method it is unnecessary to modify the smoother in any way. The restriction and prolongation operators are chosen to keep the inter-level boundary points constant during the fine grid smoothing. The complete solution over the whole domain is held implicitly over all of the grid levels. When the MLAT method is formulated as a defect correction scheme the fine grid smooths are thought of as an improvement to the accuracy of the coarse grid solve.

More recently, the Fast Adaptive Composite grid scheme (FAC) has been developed, mainly by McCormick e.g. [66]. Convergence theory for this method is given in [64] and it has been developed and applied along with other collaborators, e.g. [44, 65]. A level in the grid hierarchy is defined in a similar way to the MLAT method but the discretisation operations are also defined on a 'composite' grid which encompasses the whole domain. The 'composite' grid is composed of the finest existing elements in each subdomain and is therefore non-uniform. All smoothing is performed on uniform grids and subgrids. However the results are collected explicitly on the composite grid and the residuals are transferred from the composite grid to the level subgrids in order to calculate corrections. The composite grid stencils for the points on the inter-level boundaries are used to resolve

the residuals for these points which would otherwise be inconsistent. The inter-grid transfer functions are chosen not to interfere with the residual and solution transfer.

A 'multilevel adaptive iterative method' was introduced by Rude [82]. This technique employs a method of 'virtual global refinement' where, although all uniform grid points are 'virtually' present, only the set of 'active' points are involved in the smoothing process. A point is removed from the active set if the correction calculated for that point is below a particular tolerance. Therefore, as the smoothing iteration proceeds the active set gets smaller. When a point is corrected its neighbours are added to the active set as their residuals will have been changed. When applied as the smoother of a multigrid method this process is algebraically equivalent to the FAC method. (However it differs from the FAC method in that the grid nodes are 'activated' and 'de-activated' on an individual basis rather than being grouped into 'patches.') The method was extended in [81] to include adaptive cycling, providing more flexibility than a fixed V- or W-cycle.

All of the three methods described above avoid dealing with hanging nodes in the smoothing process either through the choice of gridding technique or through the isolation of subdomains where the smoothing is applied. The PLTMG package [9] for the solution of elliptic PDEs also avoids the issue of hanging nodes by using triangular grids with green refinement. However the FEARS package, due to Rheinboldt *et al* [98], allows the presence of hanging nodes in the grid and constrains their value using the method described in the previous section.

Multigrid applications commonly employ tree data structures as they contain an inherent definition of the level of an element in the grid. For example a 'triangle tree' is used in the PLTMG package, where the level of a triangular element corresponds to its level in the tree data structure. In [79] Rivara presents an alternative data structure for adaptive multi-

grid over conforming triangular grids based on the ‘molecular list’ structure described in Section 2.5.2. A molecular list is created for each of a sequence of nested triangulations.

Chapter 3

Applications Background

3.1 Introduction

The purpose of this chapter is to introduce the partial differential equations which are used as test problems throughout this work. For each example we give the governing equations and describe the nature and applications of the model.

The chapter is divided into three further sections. In Section 3.2 we present an example of a nonlinear elliptic equation. Section 3.3 presents two nonlinear parabolic equations and the final Section, 3.4, introduces two systems of nonlinear parabolic equations including our most demanding application based upon a phase field model for solidification.

3.2 Elliptic Partial Differential Equations

A nonlinear elliptic test problem taken from Briggs [19], is characterised by,

$$-\overrightarrow{\nabla}^2 u + \gamma u e^u = f,$$

solved over a $(-1,1) \times (-1,1)$ domain with Dirichlet conditions on the entire boundary. This is an artificial problem for which the equation has no obvious physical meaning however it may be used to test the ability of our numerical methods to deal with various levels of nonlinearity. In this work we formulate this equation to have an exact solution. This enables us to use it as an initial test case for our proposed adaptive multigrid method. The right-hand side, f, is chosen to permit $u = 1 - tanh(k(x^2 + y^2 - \alpha^2))$. This creates a solution with an area of high curvature in the region of $x^2 + y^2 = \alpha^2$.

3.3 Parabolic Partial Differential Equations

Nonlinear parabolic equations are the main interest of this thesis. In this section we present the three examples of this type of equation (although the first is linear) which have all been used in the development and testing of the proposed adaptive multigrid method. Firstly, in Section 3.3.1 there is a nonlinear diffusion problem: the porous medium equation (PME). And secondly, there is a different nonlinear parabolic problem in the form of a convection-diffusion equation in Section 3.3.2.

3.3.1 The Porous Medium Equation

The porous medium equation (PME) is a nonlinear equation that has been used to describe various time-dependent nonlinear diffusion processes [29]. For example the dispersion of

gas through a porous medium [28]. The form of the PME is

$$\frac{\partial u}{\partial t} = \overrightarrow{\nabla} \cdot (u^m \overrightarrow{\nabla} u), \tag{3.1}$$

where $u = 0$ everywhere on the Dirichlet boundary of the $(-2,2) \times (-2,2)$ domain. This is accompanied by initial conditions dictating a high concentration (u) in a small area in the centre of the domain with a concentration (u) of zero in the rest of the domain. As we take steps through time the support of u will increase in size and the value of u at the centre will decrease. The concentration level can drop sharply to zero in the contact region making this area difficult to resolve numerically. Clearly if we choose $m = 0$ the PME reduces to the heat equation. However, the higher the value chosen for m the more nonlinear the problem becomes. Again this solution has the advantage of possessing known analytical solutions in certain cases. These are radially symmetric problems whereby a solution, in two dimensions is given by:

$$s(r,t) = \begin{cases} \frac{1}{(\lambda(t))^d}\left(1 - \left(\frac{r}{r_0\lambda(t)}\right)^2\right)^{\frac{1}{m}} & r \leq r_0\lambda(t) \\ 0 & r > r_0\lambda(t) \end{cases},$$

where d is the dimension of the problem and r_0 is the radius of the initial region of non-zero concentration. (A precise definition of $\lambda(t)$ is given in Chapter 6.) We chose initial conditions to be consistent with this similarity solution for the purposes of testing.

3.3.2 A Nonlinear Convection-Diffusion Equation

We will also consider the more general parabolic form:

$$\frac{\partial u}{\partial t} + \overrightarrow{f} \cdot \overrightarrow{\nabla} u = \delta \overrightarrow{\nabla}^2 u,$$

where $\overrightarrow{f} \neq 0$, $\delta > 0$ and $u = 0$ everywhere on the Dirichlet boundary. The problem is solved over a $(-1,1) \times (-1,1)$ domain. In the case where $\overrightarrow{f} = \overrightarrow{f}(\overrightarrow{x})$ we have a linear parabolic equation and in the case where $\overrightarrow{f} = \overrightarrow{f}(\overrightarrow{x},u)$ the equation will be nonlinear. (Note that now the nonlinearity is in the convection term as opposed to the diffusion term for the PME.) Taking $f = u$ in the one-dimensional case gives us the standard one-dimensional Burgers' equation [23],

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \delta \frac{\partial^2 u}{\partial x^2}.$$

Some possibilities for the solution of this equation are analysed in [23] by Burgers. The solutions of this equation typically form shocks, the sharpness of which are controlled by the viscosity parameter, δ. This equation is often used to simulate waves. One such solution takes the form:

$$u(x,t) = \frac{1}{1+e^{(x-0.3-0.5t)/\delta}},$$

with appropriate boundary conditions. This particular solution corresponds to a wave propagating away from $x = 0$ (starting at $x = 0.3$ and moving with velocity 0.5). We would like to extend this to a two-dimensional solution which propagates radially away from the origin. Generalising the one-dimensional equation to two dimensions gives:

$$\frac{\partial u}{\partial t} + \frac{u}{r}\overrightarrow{x} \cdot \overrightarrow{\nabla} u = \delta \overrightarrow{\nabla}^2 u, \tag{3.2}$$

where $r = \sqrt{x^2+y^2}$. In this case we no longer have a similarity solution (since $\overrightarrow{\nabla}^2 u \neq \frac{\partial^2 u}{\partial r^2}$) but an approximation which is close to the true solution for small values of δ:

$$u(\overrightarrow{x},t) = \frac{1}{1+e^{(r-0.3-0.5t)/\delta}}. \tag{3.3}$$

We can create a set of initial conditions by simply setting $t = 0$ in Equation (3.3). The solution is then equal to one at the origin and anywhere behind the wave and equal to zero

in front of the waves and on the boundaries of the domain.

As before the parameter δ determines the steepness of the transition between the $u = 0$ region and the $u = 1$ region.

3.4 Systems of Parabolic Differential Equations

Many physically interesting problems, such as the Navier-Stokes equations [90] for example are described by a system of elliptic or parabolic equations. In this section we introduce two such parabolic systems. Firstly, we construct a somewhat artificial system based upon the Burgers' equation presented above. This provides us with a system of nonlinear parabolic PDEs for which we can test our methods before applying them to a more complex application. The application used here is an anisotropic phase field (PF) model for the rapid solidification of a pure melt. This is described in some detail in Section 3.4.2.

3.4.1 Systems of Convection-Diffusion Equations

An artificial system of parabolic PDEs is developed to assess the performance of the numerical solution methods. Both equations come from the nonlinear Burgers' equations as used in Section 3.3.2:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{-v}{r}(\overrightarrow{x} \cdot \overrightarrow{\nabla} u) + \delta \overrightarrow{\nabla}^2 u, \\ \frac{\partial v}{\partial t} &= \frac{u}{r}(\overrightarrow{x} \cdot \overrightarrow{\nabla} v) + \delta \overrightarrow{\nabla}^2 v. \end{aligned} \tag{3.4}$$

For small δ an approximate solution is therefore given by:

$$u(\overrightarrow{x},t) = -v(\overrightarrow{x},t) = \frac{1}{1+e^{(r-0.3+0.5t)/\delta}}.$$

The initial and boundary conditions are set up in a similar way to those of the single Burgers' equation in Section 3.3.2. The two solutions of this model are mirror images of each other with the *x*-*y* plane as the plane of symmetry.

3.4.2 Phase Field Model of Rapid Solidification

Phase field (PF) models are used, amongst other applications, to model the formation of solid structures from molten metals under cooling [49]. In such an application the final structure of the crystal depends on aspects such as speed of solidification and movement in the melt. The resulting structure of the metal affects properties such as strength, durability and electrical conductivity, making the cooling process of significant industrial interest.

The PF model was introduced for boundary evolution problems as a less complicated alternative to the front tracking method [21] which requires a parameterisation of the boundary at each time step. The simplest PF models have two dependent variables: a physical variable (e.g. temperature [50, 97]) and the phase field variable. The phase field variable describes the phase (liquid / solid) of each point in the domain, typically taking a value of 0 in the solid region and 1 in the liquid region. The PF model includes a diffuse boundary (or 'interface') between the solid and liquid regions (e.g. [12]) which simulates the change from one material phase to the next more realistically than a sharp interface. In this region the phase variable adopts an intermediate value. A simulation is typically started with a seed crystal in the melt represented by a small region where the phase variable is zero with a smaller interface region surrounding it. As the simulation progresses the solid region grows and the interface lengthens.

There are several incarnations of the PF model developed by different research groups. One model, central to the field of research, was developed by Karma and Rappel and is presented in [52]. However the formulation used here is taken from [97] by Wheeler *et al*.

Although it is possible to create computational models for metal alloys, for example [4], the formulation used here is for a pure metal. Wheeler's model is formulated for the rapid solidification of an undercooled pure melt. (Parameters corresponding to nickel are used in this work.) This fast solidification can produce highly anisotropic crystal formations (called dendrites) which are qualitatively reproduced by the PF model. In order to produce these realistic growth effects the model used incorporates anisotropy into surface tension. The effect of the level of anisotropy on the crystal formation is analysed by Kobayashi in [55].

The coupled pair of nonlinear parabolic PDEs which govern the phase field behaviour in Wheeler's model are:

$$\frac{\varepsilon^2}{m}\frac{\partial\phi}{\partial t} = r(\phi,u) + \varepsilon^2\overrightarrow{\nabla}\cdot(s(\phi)\overrightarrow{\nabla}\phi) - \varepsilon^2\frac{\partial}{\partial x}\left(v(\phi)\frac{\partial\phi}{\partial y}\right) + \varepsilon^2\frac{\partial}{\partial y}\left(v(\phi)\frac{\partial\phi}{\partial x}\right), \qquad (3.5)$$

$$\frac{\partial u}{\partial t} + \frac{1}{\triangle}P(\phi)\frac{\partial\phi}{\partial t} = \overrightarrow{\nabla}^2 u, \qquad (3.6)$$

where, $P(\phi) = \phi^3(10 - 15\phi + 6\phi^2)$. Here ϕ is the phase field variable and u is the temperature variable. Details of the precise forms of r, s and v, as well as the boundary and initial conditions, are given in Section 4.4.1

High memory use and long execution time are typical computational problems for the PF model. There is a large difference in the scale in which the PF variable is changing and the scale in which the temperature variable is changing. In order to resolve the phase variable accurately extremely fine meshes must be used. (The temperature variable requires a significantly less refined mesh.) The interface region, which is defined by the rapid change of the phase variable, is generally $\mathcal{O}(\varepsilon)$. In order to resolve the interface

sufficiently $\mathcal{O}(10)$ points would be required across its width. The grid spacing in the interface region must therefore be $\mathcal{O}(\frac{\varepsilon}{10})$. This refinement restriction makes the simulations computationally intensive, especially when combined with a conditionally stable time-stepping strategy which requires $\frac{\Delta t}{\Delta x^2}$ to remain bounded as $\Delta x \to 0$. Furthermore, PF models are becoming more sophisticated and increasingly are performed with non-pure-melts [4], or in three dimensions [47, 49, 51, 85]. Simulations in three dimensions are necessary for comprehensive and accurate comparison with experimental data. In three dimensions the computational work and storage required is intractable for sufficiently fine, uniform grids. Spatial adaptivity has therefore been introduced to reduce both of these limiting factors, e.g. Provatas *et al* in [75]. In this thesis we restrict our attention to improving computational methods in two dimensions however the issues associated with porting these improvements to three dimensions are discussed in Chapter 8.

Chapter 4

Grid Adaptivity

4.1 Introduction

This chapter details the h-refinement scheme developed in this work. The data structures and main algorithms for the scheme are given, along with the supporting techniques for grid traversal and re-adaption. Section 4.2 describes the hierarchical data structure used. In Section 4.3 we go on to show the rules and algorithms for controlling refinement and coarsening in the context of this data structure. The resulting grids have quadrilateral elements and contain 'disconnected' (or 'hanging') nodes. Finally, in Section 4.4 the gridding scheme is applied to a semi-implicit discretisation of the PF model which was introduced in Chapter 3. A modified conjugate gradient method due to Meyer [67] is used as the iterative solver in order to resolve the hanging nodes in the mesh.

4.2 Data Structures

When creating data structures to facilitate an adaptive gridding scheme we must consider how they will be used in relation to the discretisation scheme. In a typical FE discretisation implementation there is a sweep over the elements in order to build the algebraic system. There must, therefore, be a clear method for traversing the elements in the grid. An integral will be performed over each element so it must also be possible to access all the information about that element including the values at its vertices.

When dealing with uniform grids it is straight forward to set up an indexing system and a set of arrays to store the grid point values (e.g. location, solution values, etc). Finding a point within the domain, and those in its near vicinity, is then simply a case of manipulating the indices. However, this system is much less practical if the grid is non-uniform and/or regularly changing. The data structures developed in this work are intended to be practical for highly non-uniform meshes which are re-adapted continuously throughout the execution of a time-dependent simulation. In this case a flexible, dynamic data structure is needed, which accurately reflects the form of the grid and facilitates re-adaption.

A review of existing adaptive methods and the data structures used in their implementation is given in Section 2.5. We have chosen to implement a hierarchical, tree data structure where the mesh is built element-wise. Some basics of tree data structures in the C programming language can be found in [34]. It is usual to refer to items stored in a tree as ‘nodes’. However, in this work, these will be referred to as ‘elements’ as that is what they represent in the FE sense. We start with one element, covering the whole domain, which corresponds to the root element of the tree. (If the domain is more complex than a single rectangle then we would need the root of the tree to contain more than a single element.) When completed the leaf elements of the tree correspond to the active elements in the computational grid. The computational grid, in this work, is composed of square

elements, which when refined, are subdivided into four squares each with a quarter of the area of the original square. In light of this, the tree structure used is a quadtree, where each tree element has four (potential) child elements. The use of square elements imposes limits on the shape of the domain used. However the structures that we present in this chapter can be modified for use with other element shapes, such as triangles, to offer much more flexibility. This is discussed briefly in Chapter 8. The exact nature of the quadtree in this work is described below.

4.2.1 The Quadtree

The quadtree data structure is a standard tree structure, evolving from a root node. The quadtree developed here is similar to that used by Strain in [88]. In order to discuss the data structure we must establish a set of terms and their definitions. We impose the idea of 'levels' on the quadtree where: the root element is at level 0 (in this work there are never any other elements at level 0); the elements connected to the root through only one link are at level 1; those that are connected, through only one link, to level 1 elements (and have not already been assigned a level) are level 2; etc. (What is referred to here as a 'link' would typically be referred to as an 'edge' in the language of graph theory. However 'link' is used to avoid ambiguity with the meaning that 'edge' takes in FE theory.) The idea of levels gives us a framework with which to talk about the location of an element in the tree. For an element on level $\ell > 0$ we define its 'parent' element to be the unique element at $\ell - 1$ to which it is connected. (The root element has no parent.) An element can also have 'child' elements which are those directly connected to it and are on level $\ell + 1$. Those elements which have no child elements are called 'leaf' elements. The 'siblings' of an element are defined as the set of three elements at level ℓ which share a parent with the element. For this application we require a 'strict' tree, where each element has either no child elements or all four. This is in order to keep the element shapes in our grid consistent. Each node of the tree represents an element in the grid, some

active and some inactive (i.e. they have been refined). An active element is one which is involved in the solution process which, in general, corresponds to the leaf elements of the tree. As the implementation of these structures was done in the C programming language we have access to explicit pointer types and dynamic memory allocation. The structure representing an element contains the following items:

- A pointer to the parent element,
- Four pointers to the child elements,
- Four pointers to the grid points at each of the element vertices,
- The area of the element,
- An identification number,
- Manipulation flags.

The 'manipulation flags' are binary flags used during creation and re-adaption of the grid. One example of their use happens during refinement of the grid when an element can be marked for refinement (following a test based on an error estimator). Figure 4.1 illustrates the correspondence between the quadtree nodes and the elements in the grid, at various stages of development. The identification number of each element is shown in the mesh and in the tree. The structure and purpose of these identification numbers are described in Section 4.2.3.

4.2.2 The Complete Grid Data Structure

Although an element orientated structure makes sense when using FE methods, we still need to store point-wise values at the nodes of the grid. It would not make sense to do this within the quadtree as each grid point is shared between four elements so the storage and, to a lesser extent, computational time would be wasted in storing each value four

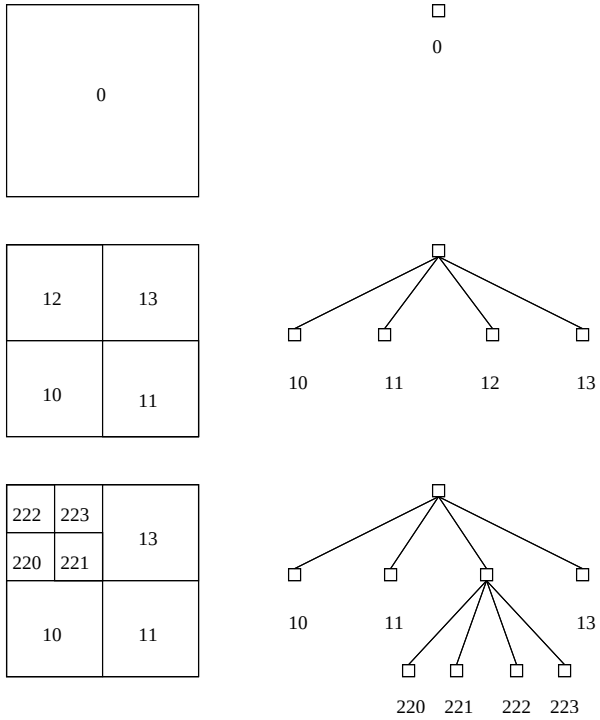


Figure 4.1: Relation of the quadtree data structure to the computational grid: Three level of the grid are shown with the corresponding quadtree. The identification numbers of each element are given.

times. Therefore, a separate list of grid points (with no repetitions) is created, which can be accessed via the elements in the tree. The point list is a dynamic, linked list to facilitate flexibility when the grid is changed. The items included in a grid point structure are:

- Solution value(s),
- Grid point co-ordinates,
- A pointer to the next node in list,
- Manipulation flags.

Again, the ‘manipulation flags’ are binary flags used during creation and re-adaption of the grid so each point can be, for example, marked as a newly created point. The full data structure setup is illustrated in Figure 4.2 where the grid now shows the number system of the grid points corresponding to the nodes of the linked list.

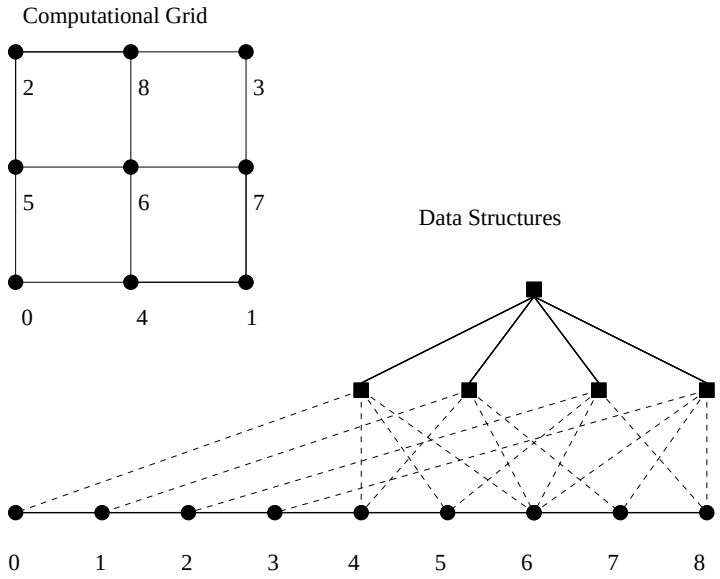


Figure 4.2: Relation of quadtree to point list: The quadtree is represented, as in previous figures, with squares as elements. The links are shown between these elements and the points (at their vertices) stored in the point list. The data structure applies to the simple grid above.

4.2.3 Traversing the Grid

In order to effectively use the grid structure that we have created, we need techniques for traversing the element tree in an efficient manner and accessing the particular elements efficiently. There are two main challenges to overcome: to traverse all the leaf elements in the tree efficiently; and to find the neighbouring elements of any element in the grid. These tasks are tackled in the following sections.

Efficient Traversal

In order to solve the FE problem on our computational grid it must be possible to traverse the leaf elements of the quadtree as typical FE codes have an outer loop over the elements. One method of facilitating the traversal of the active elements in the grid is to set up a linked list of all of these active elements as in [75]. Once this is done the traversal of the active elements is trivial. However there are overheads associated with this method in terms of storage and computational work. An alternative is to create algorithms which

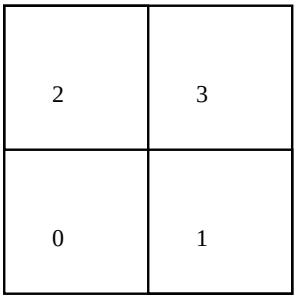


Figure 4.3: Ordering of elements in the grid: This is the ordering chosen for the child elements of any parent element in the quadtree.

will perform the traversal without the need for more pointers in the structure. As we would like to use this data structure for time-dependent problems where the mesh refinement, and so the arrangement of active elements, will change throughout the execution, the second approach was chosen.

A depth-first traversal was adopted, which uses the element pointers and an ordering imposed on the tree, to move from one leaf element to the 'next' one. Numbering each set of sibling elements from 0 to 3 imposes a natural ordering on the tree. The order was chosen arbitrarily and can be seen in Figure 4.3. The tree itself is navigated using a structure containing two pointers, one to the root element and one to the 'current' element. The latter is essential for the traversal algorithms. Moving from one element to another during a traversal involves changing the assignment of the current pointer. Similarly, moving to an arbitrary element means giving the current pointer the address of this structure. The traversal of the active (or leaf) elements involves two main methods, one to set the current element to be the first leaf element in the tree and a second, recursive, method to move the current element to be the next leaf element, according to the ordering. Locating the first leaf element is straight forward. Starting at the root we choose child 0 of each element until we reach an element with no children. The algorithm for moving from one leaf element to the next proves to be a little more complex. This is given in Algorithm 3. Below are three examples of how the algorithm works:

Algorithm 3 Function: Goto Next Leaf Element

```
1:  Input parameter: C - pointer to current element
2:  if C != Root Element of Tree then
3:      Set V = C
4:      Set C = C.parent
5:      if  V == C.child[0] then
6:          C = C.child[1]
7:          while C is not a leaf element do
8:              C = C.child[0]
9:          end while
10:     else if V == C.child[1] then
11:         C = C.child[2]
12:         while C is not a leaf element do
13:             C = C.child[0]
14:         end while
15:     else if V == C.child[2] then
16:         C = C.child[3]
17:         while C is not a leaf element do
18:             C = C.child[0]
19:         end while
20:     else
21:         Call 'Goto Next Leaf Element' with C
22:     end if
23: else
24:     Indicate that the last leaf element has been reached.
25: end if
```

- If element *[1,1]* in Figure 4.1 was given as element C in Algorithm 3 the second conditional statement would be true (i.e. $V == C.child[1]$). Element *[1, 2]* would be assigned to C (this is not labelled in the figure), then the while loop would execute once to assign element *[2, 2, 0]* to C.

- If element *[2, 2, 3]* from Figure 4.1 is given as the initial C. Recursion would be used to recall the function with the parent of element *[2, 2, 3]* (i.e. element *[1, 2]*). In this new instance of the method the third condition is used and element *[1, 3]* is assigned to C. As the current element C is passed by reference the changes made during recursion are maintained after both instances of the method have finished.

- If element *[1, 3]* is initially assigned to C. Recursion would be used to recall the function with the root element. This second instance would recognise that the final leaf element had been reached.

Finding the Neighbour of an Element

In order to check the validity of the grid during the refinement and de-refinement procedures (described in detail in Section 4.3) we need to be able to identify the neighbours of any given element. We approach this task by first creating a method with which to set up a pointer to any given element in the grid. Each element is given a unique identification number (ID) which allows us to locate the element to which we wish to establish a pointer. These IDs are arrays of integers which reflect the position of the element in the tree. The first entry corresponds to the level of the tree where the element resides and the rest reflect the positioning of the element. In Figure 4.1 we can see that the element with identification number *[2, 2, 0]* is on level 2, within the area covered by the 2nd child of the root node and is child 0 of its parent. The ID, therefore, traces the ancestry of an element starting at the root element. It is straightforward to set a pointer to any element in the tree provided that we have its ID. The simple algorithm for finding an element using

its ID is given in Algorithm 4.

Algorithm 4 Find Element

```
1: C = Root Element
2: for i = 1 to ID[0] do
3:     C = C.child[ID[i]]
4: end for
```

In building and re-adapting the grid, as well as in some solution processes, it is necessary to know which elements border the current one. The elements which border an element are termed its 'neighbours'. We divide these into two types: those which share an edge with the element, called 'primary neighbours'; and those which only share a point with the element, called 'secondary neighbours'. There is no direct method of finding the neighbours of an element inherent in the quadtree structure. In fact, it can prove to be difficult as two elements, next to each other in the grid, can be on separate branches of the tree reaching back to the root. An efficient algorithm to identify the neighbours of an element can make use of the ID numbers defined above. Rather than attempting to move a pointer through the tree, from one element to its neighbour, we use the element's ID and the relative position of the neighbour to calculate the ID of the neighbour. The neighbour's ID can then be used to set a pointer to the neighbour using the Algorithm 4. We can establish from the last digit of the element's ID which neighbours are the siblings of the element and which belong to other branches of the tree.

An example of this is given in Figure 4.4. The element E has the ID *[3, 0, 3, 3]* and hence the X- and Y- neighbours are simple to find, as they are siblings of E. The IDs of the other two primary neighbours, X+ and Y+ are found by using the same principle used recursively up the tree. The algorithms for finding the ID of the X+ neighbour is given in Algorithm 5. The algorithm finds the ID of the theoretical, X+ neighbour which would be on the same grid level as the element. At this stage it does not matter whether this element

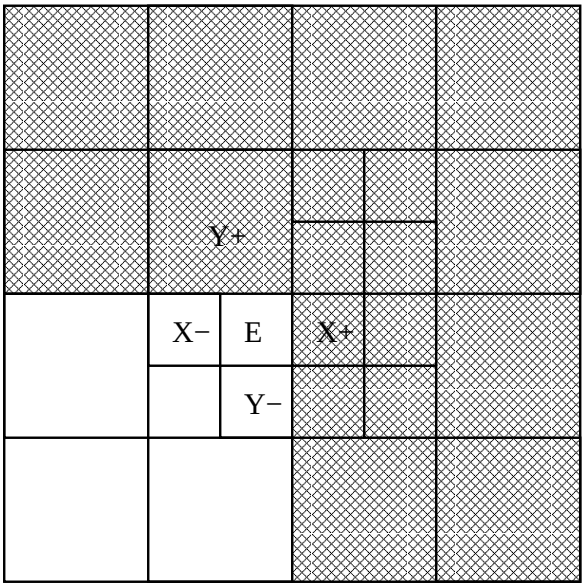


Figure 4.4: The primary neighbours of a quadrilateral element: The four primary neighbours of the element marked E are marked X+, Y+, X- and Y- according to their relative orientation.

actually exists in the tree. We can then use this ID to set a pointer to the neighbouring element, even if this turns out to be an ancestor of the element characterised by the ID. In the case of element E in Figure 4.4 for the X+ neighbour (or the element directly to the right of E) we can trace the steps of Algorithm 5 as follows.

- When the method is initially called: $XP.ID = [3,0,3,3]$ and $\ell = 3$. The method will then be called recursively.
- In the second instance of the method: $XP.ID = [3,0,3]$ and $\ell = 2$. Again the method is called recursively.
- In the third instance of the method: $XP.ID = [3,0]$ and $\ell = 1$. In this case XP.ID is changed to *[3, 1]* and the method instance terminates.
- On returning to the second instance of the method XP.ID is changed to *[3, 1, 2]* and this instance of the method terminates.
- On returning to the initial instance of the method XP.ID is changed to *[3, 1, 2, 2]*

Algorithm 5 Function: Find ID of X-Plus Neighbour of Element

```
1: Input parameter: E.ID - identification number of current element
2: Input parameter: XP.ID - identification number of neighbour directly to the right of
   the current element. (Initially empty.)
3: ℓ - level of current element
4: Set XP.ID = E.ID
5: if XP.ID[0] == 0 then
6:    We are at the root so there are no neighbours
7:    Set XP.ID[0] = -1
8: else
9:    if XP.ID[ℓ] == 0 then
10:      XP.ID[ℓ] = 1
11:   else if XP.ID[ℓ] == 1 then
12:      Call 'Find ID of X-Plus Neighbour of Element' with E.ID, XP.ID and ℓ − 1
13:      XP.ID[ℓ] = 0
14:   else if XP.ID[ℓ] == 2 then
15:      XP.ID[ℓ] = 3
16:   else if XP.ID[ℓ] == 3 then
17:      Call 'Find ID of X-Plus Neighbour of Element' with E.ID, XP.ID and ℓ − 1
18:      XP.ID[ℓ] = 2
19:   end if
20: end if
```

and this remaining instance terminates.

In the case where the element borders on the boundary of the domain and a neighbour does not exist the method will be called recursively until the current element is the root element. At this point the final instance will recognise that there are no neighbours and pass this information back through a -1 value in $XP.ID[0]$. The algorithms for finding the X-, Y+ and Y- neighbours are similar to Algorithm 5 for the X+ neighbour.

The secondary neighbours are found by using a combination of the methods for finding the primary neighbours. For example, the secondary neighbour to the top right of element E would be located in two stages. Firstly the X+ neighbour of E would be found and a pointer would be established to this element. The algorithm for finding the Y+ neighbour of an element would then be used with the newly found X+ neighbour element as an input.

4.3 Adapting the Grid

In this section the method for refining and coarsening the grid is described in detail. The conditions under which refinement and coarsening occur are discussed and the algorithms for implementing these are given. The mechanisms for creating and removing elements in the grid are also described.

Firstly, it is necessary to understand a little more about the computational grid. There are (user defined) maximum and minimum grid levels. The level of an element in the grid is equivalent to its level in the quadtree structure, so a 'grid level', ℓ, defines all of the elements of level ℓ. The grid is developed in a globally refined manner up to the minimum grid level. (The re-adaption methods then prevent any element from being coarsened beyond this level.) The maximum grid level serves as a restriction on the refinement. No elements can be created that are smaller than those at the maximum level. In order to ensure a robust solution we must preserve some smoothness in the mesh. It is therefore required that each neighbouring pair of elements only differs in area by a factor of four (or by one refinement level). This is referred to here as the validity of the mesh. It is assumed that some form of error estimator is used to drive the adaptivity of the mesh. Here we simply require that this estimator provides a numeric value for each element in the grid which in some way represents the error over that area.

In order to guarantee that the solution is sufficiently resolved, extra layers of finest level elements are added to those created by the error estimator driven refinement. The number of safety layers of refinement can be controlled by the user. Adding a safety layer to the refinement means the refinement of all of the elements in the grid which border finest level elements but are not finest level elements themselves. For time-dependent problems, where the area of high activity will move throughout the solution process, these extra layers of refinement help to ensure that the solution remains sufficiently resolved. In Section

4.4.4 we look at the balance to be found between the guarantee of a good solution and the extra expense required to solve for these extra elements.

4.3.1 Refinement

The refinement process is split into two halves, a marking stage and a refinement stage, similar to the methods in [59]. The marking stage uses a given error estimate to flag those elements which are in areas requiring further refinement. This involves one sweep over the active elements where, on each element, the error estimate is compared to a user-selected tolerance, appropriate for the problem being solved. One of the binary flags of that element is then changed to indicate whether it requires further refinement. A second sweep of the elements is made and elements bordering on those already marked for refinement are marked also. This forms the first layer of safety refinement. Any number of safety layers of refinement can be added by further sweeps if desired.
The process of refining the elements, based on the way they have been marked, is more complex than the marking as it also bears the responsibility of preserving the validity of the mesh. The validity of the mesh is ensured by following a set of rules for refinement and a set for coarsening. The two refinement rules are:

- An element cannot be further refined once it is at the maximum level.
- Two neighbouring elements should differ by at most one refinement level.

In a situation where the refinement of an element would invalidate the mesh the algorithm outlined in Algorithm 6 recursively performs any necessary refinement of neighbouring elements before refining the given element. The use of this recursive algorithm will cause the refinement of elements that have not been marked for refinement. However the extra computational work caused by these elements is justified in order to maintain a valid grid. The recursive checking of neighbouring elements could be done in the marking stage rather than the refinement stage (as in the coarsening algorithms below). However

Algorithm 6 Function: Valid Refinement

```
1: Input parameter: E - element to be refined
2: if E.level != max level then
3:    for All neighbouring elements, N do
4:       if N is coarser than E then
5:          Call 'Valid Refinement' with N
6:       end if
7:    end for
8: end if
9: Refine E
```

it would require the same amount of computational work so the choice is arbitrary.

We have, so far, referred to the element refinement but have not discussed the details of how it is implemented with respect to the data structures. This is straightforward for the elements and slightly more complex for the associated grid points. In order to refine an element, E, we take the following steps:

- Create four child elements, each with a quarter of the area of E.
- For each of the new elements:
  - Set the child pointers to NULL
  - Create the ID (using the E.ID and the tree ordering)
  - Set the parent pointer to E
  - Set the relevant child pointer of E
- Link the new elements to any of their grid points which already exist in the point list. The points that are shared with E always exist in the list. The primary neighbours of E are checked. If these neighbours have children, then points will exist along the border of E. Figure 4.5 shows two possible configurations.
- Create the remaining grid points and give them values interpolated from surrounding points. The point shared between all four new elements must always be created.

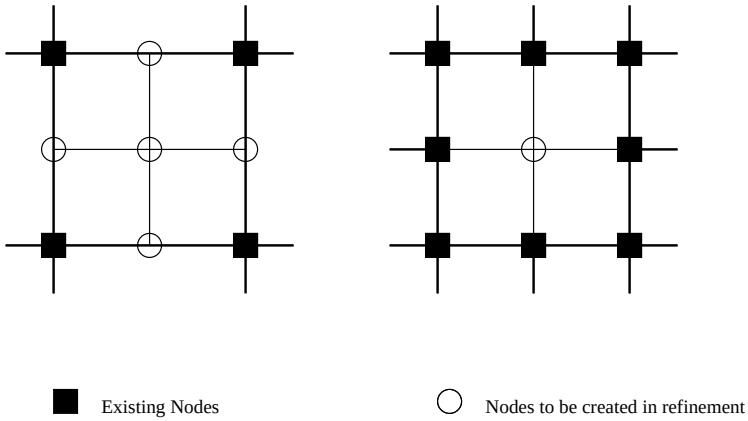


Figure 4.5: Creating points during refinement: In the left-hand case five points must be created as none of the parents neighbours are refined. In the right-hand case all of these neighbours are refined and so only one point need be created.

- Add the new points into the linked list of grid points and resolve the pointers from the new elements.

4.3.2 Coarsening

In order to maintain simplicity in the grid adaption the only coarsening that is allowed is the reverse of the refinement technique. A set of four sibling elements are removed and their parent element is made active in the computational grid. We will not allow the merging of elements which do not share the same parent.

The coarsening methods are divided into marking operations and coarsening operation, in a similar way to the refinement scheme. The validity of the mesh is taken care of in the marking scheme rather than with a recursive coarsening function. It follows that the complexity lies in the element marking scheme rather than the coarsening itself. All elements in the grid are marked to be coarsened unless:

1. They are already at the minimum level,

2. They (or any of their siblings) exceed the error tolerance,

3. They (or any of their siblings) have child elements,

4. Any of the neighbours of their parent element have grandchildren.

Clearly we wish to prevent the coarsening from undoing the work of the refinement. Item 2 (above) ensures that an element marked for refinement is not coarsened. The same error estimate is used here as in the refinement scheme however a different tolerance is used. This is discussed further in Section 4.3.3. The coarsening scheme is designed to be as cautious as possible while still being effective. For example, Item 3 is a safety measure which ensures that only one level of refinement can be removed in one sweep of the coarsening algorithm. The validity of the mesh is preserved by Item 4. When removing a set of elements and exposing their parent, all the neighbouring elements must be of the same level as those to be removed, or coarser. If any neighbouring element were finer than those to be removed the coarsening would invalidate the mesh. Using rule 4, in some cases, delays valid coarsening until the next pass of the coarsening algorithm. The algorithm could be constructed to be more aggressive but caution was favoured in order to ensure an properly resolved solution. Some examples of when coarsening is and is not permitted are given in Figure 4.6. In this figure, example 5 shows an instance where the coarsening would be valid but is prevented by rule 4.

The operation of coarsening is simpler than that of refining. A group of four sibling elements are removed from the tree making the parent a leaf element, now active in the tree. The grid point(s) which belonged solely to these elements are then removed from the linked list. The grid point shared between all four sibling elements is removed. The four grid points shared with the parent element are kept. The remaining four points on the edges of the parents are kept if the relevant neighbour is refined and removed otherwise.

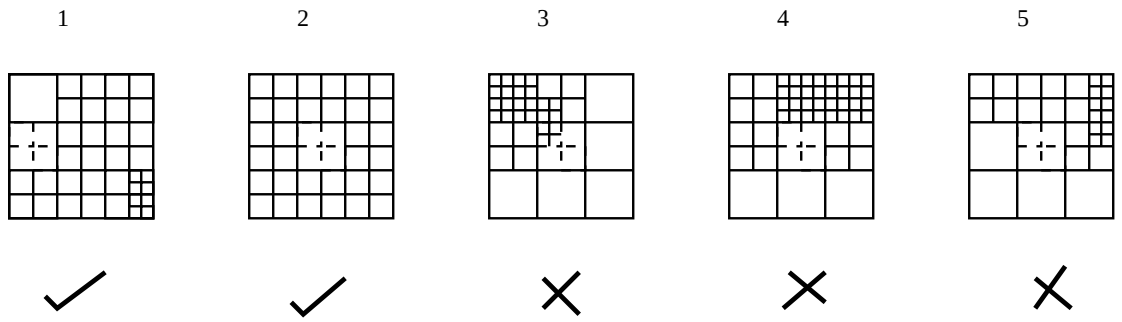


Figure 4.6: Various coarsening scenarios: The elements which are shown with dotted lines are proposed for coarsening and the tick or cross underneath each example indicates whether the coarsening rules the algorithm would allow the coarsening to take place.

4.3.3 Re-adaption

The refinement and coarsening processes described above are wrapped within an algorithm for re-adapting the grid. The refinement is performed first, followed by the coarsening. This ordering is arbitrary as the refinement and coarsening could be performed in either order, provided that they remain separate operations. The order of these operations is illustrated in Algorithm 7. In order to prevent these processes interfering with each

Algorithm 7 Grid Re-adaption

1: Mark elements for refinement
2: Perform refinement (using recursive grid correction)
3: Mark elements for coarsening
4: Perform coarsening (with no recursion)

other the error tolerance used for refinement is different from that used for coarsening. Each is based on the same error indicator but the activation tolerance is different. The refinement tolerance, $rtol$, dictates the value of the indicator above which local refinement is required, whereas the coarsening tolerance, $ctol < rtol$, is used to choose which elements may be coarsened (since the error, which is less than $ctol$, is unnecessarily small). This two-tier tolerance scheme defines a set of elements (with an error indicator value between the two tolerances) which are 'okay as they are.' These elements form a buffer region between those being refined and those being coarsened. In general, experience suggests that it is best to keep an order of magnitude difference between the two tolerance values.

4.3.4 Hanging Nodes

In Section 2.5.3 we discussed the options for the resolution of the hanging nodes in a non-uniform quadrilateral mesh including the 'red' and 'green' refinement possibilities. In the case of red refinement the discontinuous FE method could be used, where the discontinuities would be allowed to remain in the solution. Examples of the discontinuous FE method being used with grid containing hanging nodes can be found in [35, 57]. Discontinuous Galerkin methods were not attempted in this work as the focus is on simplicity of understanding and implementation and these methods are more complicated to formulate than the continuous Galerkin. In addition to this the number of degrees of freedom required for the discontinuous version of the method is substantially higher.

For the grids in this work the mesh validity criterion dictates that there is never more than one hanging node per element edge. This restriction simplifies the work of resolving the hanging nodes. (However it is possible to assemble for grids with multiple points along an edge, for example [35].) An alternative approach was used to produce results on grids containing hanging nodes based upon constraining the solution values at such nodes. This is implemented using a projection technique which is a generalisation of that proposed in [67] and full details are given in Section 4.4.3.

We maintain a record of the hanging nodes in order to identify them during the solution process. A dynamic linked list is used which is re-formed each time the mesh is changed. We define the parent points of a hanging node to be the two adjacent points belonging to the largest adjacent element. The structure stored for each hanging node includes:

- A pointer to the hanging node in the point list,
- Two pointers to the parent points in the point list,
- A pointer to the next structure in the hanging node list.

4.3.5 Error Estimation and Indication

In order to drive the adaptivity for the gridding methods presented in this thesis an error estimation or indication method is needed. The implementation requires that this error estimator can produce a value for each element in the grid which in some way represents (as the name suggests) an estimate of the error over the element area. Alternatively an error indication technique can be used, which uses some attribute of the problem or local features of the approximate solution to indicate areas where the error is likely to be high. Clearly the more sophisticated approach is to use an error estimation technique, where an approximation to the error itself is calculated, giving a more precise measure of where increased refinement is required.

Error Estimators

Error estimators fall into the two categories: *a priori* and *a posteriori*. *A priori* methods use knowledge of the problem to be solved and the nature of the solution to predict the error trends in the approximations without making explicit reference to, or use of, a computed solution. In [61] Shishkin meshes are developed using an *a priori* error estimation method. These methods are, however, not much use for spatially adaptive methods since the estimates tend to be asymptotic and do not yield information on precisely where to refine or coarsen. In contrast, *a posteriori* methods use problem data, along with the calculated approximate solution, to calculate an estimate of the error in a numerical solution once it has been found. This type of method is much more useful for adaptive applications and so will be described in a little more detail here. A much more complete review of *a posteriori* error estimation, for elliptic problems, can be found in the book by Ainsworth and Oden [2].

A popular and simple type of *a posteriori* error estimation technique is based on the recovery of solution gradients. Here a post processing step is used to recover the solution

gradient from the approximate solution. The difference between this recovered gradient and the gradient of the approximate solution is used to form an estimate of the error. The most widely used approach of this type was developed by Zienkiewicz and Zhu [100]. Although this method could be regarded as crude it can be very effective and is popular with engineers. A gradient recovery method for piecewise linear finite element methods is developed in [7] and the equivalent for piecewise bilinear finite elements can be found in [53]. The Zienkiewicz and Zhu method was later developed into a local patch recovery technique in [101, 102].

A more sophisticated error estimation technique involves calculating a solution to the residual equation for the error. Consequently this method requires an approximation scheme for the error estimation procedure itself. Bank and Weiser describe several error estimators of this type in [10]. Their methods produce local error estimates by solving Neumann boundary problems over each element.

Error estimators for elliptic problems were often applied to parabolic problems on the assumption that the temporal error is maintained at a significantly lower level than the spatial error. However, in recent years *a posteriori* methods have been extended for use with parabolic problems. One example of these extended methods is described in [74].

Error Indicators

As stated above error indication techniques use features of the approximate solution to identify areas where the error is likely to be high. One of the possible indicator quantities is the gradient of the approximate solution. For certain problems areas of high gradient coincide with areas which require high grid resolution. In other cases the curvature of the approximate solution may be a more appropriate quantity for indicating areas of high error. Although these two are the simplest and therefore the most commonly used

indicators, others can be used. For example the curl of the solution, which will give an indication of the level of vorticity in a vector field solution. Where systems of differential equations are solved a decision must be made as to which dependent variable provides the most suitable choice to be used for the error indicator. Alternatively, the indication value can be calculated for each independent variable in the system and some linear combination of the resulting values can be used. The weightings within this linear combination will be dictated by the importance of the respective variables. An example of the use of this method can be seen in [73].

The grid adaptivity methods developed in this work are designed in such a way as to allow the use of any error indicator or estimation technique provided that it will produce a value for each element in the grid. As the subject of error estimation is not the focus of this work, a simple error indication method is used. The gradient of the approximate solution is calculated on each element in the grid and it is assumed that the areas of high gradient are those that require a higher level of refinement. This method has proved to be sufficient for the purposes of testing the grid adaptivity and solution methods in this work. However, the implementation is flexible enough to function equally well if a more sophisticated approach is used.

4.4 Application of the Adaptive Mesh Algorithm

In this section we demonstrate the correctness of our implementation of the mesh adaptivity by applying it to the solution of the nonlinear PF model, which was introduced in Section 3.4.2. Phase field models such as this have been solved previously using adaptively refined meshes: see for example the work of Provatas *et al* presented in [75]. In that paper they describe a green refinement scheme combining quadrilaterals with temporary triangular elements in the semi-implicit solution of a typical PF model. Here we

demonstrate that an equivalent scheme can be constructed where the hanging nodes are left unresolved by the refinement scheme but handled by a projected conjugate gradient solution algorithm developed by Meyer in [67]. In later chapters we go on to construct a fully implicit discretisation of the specific PF model introduced in Chapter 3 and solve this using the new multigrid solution scheme developed during this work.

4.4.1 Finite Element Discretisation in Space

Recall from Section 3.4 that the following system of partial differential equations provide an anisotropic model of rapid solidification behaviour for a pure melt:

$$\frac{\varepsilon^2}{m}\frac{\partial\phi}{\partial t} = r(\phi,u) + \varepsilon^2\overrightarrow{\nabla}\cdot(s(\phi)\overrightarrow{\nabla}\phi) - \varepsilon^2\frac{\partial}{\partial x}\left(v(\phi)\frac{\partial\phi}{\partial y}\right) + \varepsilon^2\frac{\partial}{\partial y}\left(v(\phi)\frac{\partial\phi}{\partial x}\right), \tag{4.1}$$

$$\frac{\partial u}{\partial t} + \frac{1}{\triangle}P(\phi)\frac{\partial\phi}{\partial t} = \overrightarrow{\nabla}^2 u. \tag{4.2}$$

For the purposes of this work we use the following definitions of the terms in (4.1) and (4.2), based upon [97] where

$$r(\phi,u) = \phi(1-\phi)[\phi - \frac{1}{2} + 30\varepsilon\alpha\Delta u\phi(1-\phi)] \tag{4.3}$$

$$s(\phi) = \eta^2(\theta(\phi)), \tag{4.4}$$

$$v(\phi) = \eta(\theta(\phi))\eta'(\theta(\phi)), \tag{4.5}$$

$$\eta(\theta) = 1 + \gamma cos(k\theta), \tag{4.6}$$

$$\eta'(\theta) = -\gamma k sin(k\theta), \tag{4.7}$$

and

$$P(\phi) = 30\phi^2(1-2\phi+\phi^2). \tag{4.8}$$

The PF system used here is taken directly from [97] and represents a simple model for rapid solidification in an undercooled melt. The parameter $\Delta = 0.5$ represents the undercooling of the melt. The qualitatively realistic finger-like shapes that may be obtained using this model are caused by anisotropy in the curvature of the solid-liquid interface. The level of this anisotropy is controlled by $\gamma = 0.03$. The phase field model parameters corresponding to nickel, [97], were used for all of the results appearing in this work. Further parameters include $\varepsilon = 0.005$, which corresponds to the thickness of the interface between the solid and liquid regions, and $\alpha = 400.0$ and $m = 0.05$. The parameter θ represents the angle between the solid-liquid interface normal and the x-axis of the domain.

Dirichlet conditions are used on the entirety of the boundary. The solidification simulation is initialised with a small 'seed' in the centre of the domain where the phase field variable, ϕ, takes the value of 0 representing the solid state. A continuous Galerkin finite element method with piecewise bilinear elements is used for the spatial discretisation. As in Section 2.2.1, we create a weak form of Equations (4.1) and (4.2). Then the FE trial and test functions are substituted in place of the continuous variables. Finally the spatially discretised equation systems are manipulated into an initial value ODE problem in matrix form.

The Weak Form

For the phase Equation (4.1) a suitable weak form may be derived as:

$$\frac{\varepsilon^2}{m}\int_\Omega \frac{\partial \phi}{\partial t} w d\Omega = \int_\Omega r(\phi,u) w d\Omega + \varepsilon^2 \int_\Omega \overrightarrow{\nabla}\cdot(s(\phi)\overrightarrow{\nabla}\phi) w d\Omega$$

$$-\varepsilon^2 \int_\Omega \frac{\partial}{\partial x}\left(v(\phi)\frac{\partial \phi}{\partial y}\right) w d\Omega + \varepsilon^2 \int_\Omega \frac{\partial}{\partial y}\left(v(\phi)\frac{\partial \phi}{\partial x}\right) w d\Omega. \tag{4.9}$$

From the product rule we know that

$$\overrightarrow{\nabla} \cdot (s(\phi)\overrightarrow{\nabla}\phi)w = \overrightarrow{\nabla} \cdot (ws(\phi)\overrightarrow{\nabla}\phi) - \overrightarrow{\nabla} w \cdot s(\phi)\overrightarrow{\nabla}(\phi).$$

We can therefore replace the first of the second order terms in Equation (4.9) as follows:

$$\varepsilon^2 \int_\Omega w\overrightarrow{\nabla} \cdot (s(\phi)\overrightarrow{\nabla}\phi)d\Omega = \varepsilon^2 \int_\Omega \overrightarrow{\nabla} \cdot (ws(\phi)\overrightarrow{\nabla}\phi)d\Omega - \varepsilon^2 \int_\Omega s(\phi)\overrightarrow{\nabla} w \cdot \overrightarrow{\nabla}\phi d\Omega.$$

Using the divergence theorem this becomes:

$$\varepsilon^2 \int_\Omega w\overrightarrow{\nabla} \cdot (s(\phi)\overrightarrow{\nabla}\phi)d\Omega = \varepsilon^2 \int_{\partial\Omega} \overrightarrow{\hat{n}} \cdot (ws(\phi)\overrightarrow{\nabla}\phi)ds - \varepsilon^2 \int_\Omega s(\phi)\overrightarrow{\nabla} w \cdot \overrightarrow{\nabla}\phi d\Omega,$$

where $\overrightarrow{\hat{n}}$ is the outward unit normal. Since we are assuming Dirichlet boundary conditions we restrict the test functions, w, to be zero on the boundary and so the boundary integral is zero, hence

$$\varepsilon^2 \int_\Omega w\overrightarrow{\nabla} \cdot (s(\phi)\overrightarrow{\nabla}\phi)d\Omega = -\varepsilon^2 \int_\Omega s(\phi)(\overrightarrow{\nabla}\phi \cdot \overrightarrow{\nabla} w)d\Omega.$$

The other second order terms in the phase equation and the first term in the temperature equation right-hand side are replaced in a similar manner. For the final weak form of the system it is necessary to find $\phi, u \in H^1_E(\Omega)$ such that:

$$\frac{\varepsilon^2}{m}\int_\Omega \frac{\partial\phi}{\partial t}wd\Omega = \int_\Omega r(\phi,u)wd\Omega - \varepsilon^2 \int_\Omega s(\phi)(\overrightarrow{\nabla} w \cdot \overrightarrow{\nabla}\phi)d\Omega$$

$$+\varepsilon^2 \int_\Omega \frac{\partial w}{\partial x}v(\phi)\frac{\partial\phi}{\partial y}d\Omega - \varepsilon^2 \int_\Omega \frac{\partial w}{\partial y}v(\phi)\frac{\partial\phi}{\partial x}d\Omega, \tag{4.10}$$

$$\int_\Omega \frac{\partial u}{\partial t}wd\Omega = -\int_\Omega \overrightarrow{\nabla} u \cdot \overrightarrow{\nabla} wd\Omega - \frac{1}{\triangle}\int_\Omega P(\phi)\frac{\partial\phi}{\partial t}wd\Omega, \tag{4.11}$$

holds for all $w \in H^1_0(\Omega)$, where $H^1_E(\Omega) = \{v \in H^1 : \text{v satisfies the given Dirichlet boundary conditions}\}$ and $H^1_0(\Omega) = \{v \in H^1 : v = 0 \text{ on } \partial\Omega\}$. Note that if we chose to use Neumann

boundary conditions, where the boundary condition is $\frac{\partial \phi}{\partial \hat{n}} = 0$, the boundary integrals still disappear since they are of the form $\int_{\partial\Omega} ws(\phi)\frac{\partial \phi}{\partial \hat{n}} ds$.

Spatial Discretisation

We now replace the dependent variables ϕ and u with discretised versions, constructed using a linear combination of the standard piecewise bilinear basis functions:

$$\overline{\phi} = \sum_{i=1}^{n+nb} \phi_i(t) N_i(x,y), \qquad \overline{u} = \sum_{i=1}^{n+nb} u_i(t) N_i(x,y), \tag{4.12}$$

where n is the number of unknowns, nb is the number of points on the Dirichlet (or essential) boundary and N_i is the i^{th} basis function. We are now treating $\underline{\phi}$ and $\underline{u}$ as ordered sets of time-dependent values, not dependent on space (with the unknown values ordered first and the known, Dirichlet values ordered last). The weight functions, in the weak form, (4.10) and (4.11), are replaced by N_j for $j = 1, \ldots n$ to give:

$$\frac{\varepsilon^2}{m}\int_\Omega \frac{\partial \overline{\phi}}{\partial t} N_j d\Omega = \int_\Omega r(\overline{\phi}, \overline{u}) N_j d\Omega - \varepsilon^2 \int_\Omega s(\overline{\phi})(\vec{\nabla}\overline{\phi} \cdot \vec{\nabla} N_j) d\Omega$$

$$+\varepsilon^2 \int_\Omega \frac{\partial N_j}{\partial x} v(\overline{\phi}) \frac{\partial \overline{\phi}}{\partial y} d\Omega - \varepsilon^2 \int_\Omega \frac{\partial N_j}{\partial y} v(\overline{\phi}) \frac{\partial \overline{\phi}}{\partial x} d\Omega \tag{4.13}$$

$$\int_\Omega \frac{\partial \overline{u}}{\partial t} N_j d\Omega = -\int_\Omega \vec{\nabla}\overline{u} \cdot \vec{\nabla} N_j d\Omega - \frac{1}{\triangle}\int_\Omega P(\overline{\phi}) \frac{\partial \overline{\phi}}{\partial t} N_j d\Omega. \tag{4.14}$$

We now have a system of $2n$ ordinary differential equations for $2n$ unknowns.

A Matrix Form

In the case of Dirichlet boundary conditions (which we will assume from now on) we can use the expansions (4.12) to express Equations (4.13) and (4.14) as:

$$\frac{\varepsilon^2}{m}\left[\sum_{i=1}^{n} \dot{\phi}_i \int_\Omega N_i N_j d\Omega + \sum_{i=n+1}^{n+nb} \dot{\phi}_i \int_\Omega N_i N_j d\Omega\right] = \int_\Omega r(\overline{\phi}, \overline{u}) N_j d\Omega,$$

$$-\varepsilon^2 \int_\Omega s(\overline{\phi})\,\overrightarrow{\nabla}\overline{\phi}\cdot\overrightarrow{\nabla}N_j d\Omega$$

$$+\varepsilon^2 \int_\Omega v(\overline{\phi})\frac{\partial\overline{\phi}}{\partial y}\frac{\partial N_j}{\partial x}d\Omega - \varepsilon^2 \int_\Omega v(\overline{\phi})\frac{\partial\overline{\phi}}{\partial x}\frac{\partial N_j}{\partial y}d\Omega,$$

and

$$\left[\sum_{i=1}^{n}\dot{u}_i\int_\Omega N_iN_j d\Omega + \sum_{i=n+1}^{n+nb}\dot{u}_i\int_\Omega N_iN_j d\Omega\right] = \left[-\sum_{i=0}^{n}u_i\int_\Omega \overrightarrow{\nabla}N_i\cdot\overrightarrow{\nabla}N_j d\Omega\right.$$

$$\left. -\sum_{i=n+1}^{n+nb}u_i\int_\Omega \overrightarrow{\nabla}N_i\cdot\overrightarrow{\nabla}N_j d\Omega\right] - \frac{1}{\triangle}\int_\Omega P(\overline{\phi})\dot{\overline{\phi}}N_j d\Omega, \tag{4.15}$$

for $j = 1,...,n$. The equations may be rearranged so that the unknown time derivatives are on the left-hand side and all other terms are on the right-hand side:

$$\frac{\varepsilon^2}{m}\sum_{i=0}^{n}\dot{\phi}_i\int_\Omega N_iN_j d\Omega = -\frac{\varepsilon^2}{m}\sum_{i=n+1}^{n+nb}\dot{\phi}_i\int_\Omega N_iN_j d\Omega + \int_\Omega r(\overline{\phi},\overline{u})N_j d\Omega,$$

$$-\varepsilon^2 \int_\Omega s(\overline{\phi})\,\overrightarrow{\nabla}\overline{\phi}\cdot\overrightarrow{\nabla}N_j d\Omega$$

$$+\varepsilon^2 \int_\Omega v(\overline{\phi})\frac{\partial\overline{\phi}}{\partial y}\frac{\partial N_j}{\partial x}d\Omega - \varepsilon^2 \int_\Omega v(\overline{\phi})\frac{\partial\overline{\phi}}{\partial x}\frac{\partial N_j}{\partial y}d\Omega. \tag{4.16}$$

and

$$\sum_{i=0}^{n}\dot{u}_i\int_\Omega N_iN_j d\Omega = -\sum_{i=0}^{n}u_i\int_\Omega \overrightarrow{\nabla}N_i\cdot\overrightarrow{\nabla}N_j d\Omega$$

$$-\sum_{i=n+1}^{n+nb}u_i\int_\Omega \overrightarrow{\nabla}N_i\cdot\overrightarrow{\nabla}N_j d\Omega - \sum_{i=n+1}^{n+nb}\dot{u}_i\int_\Omega N_iN_j d\Omega - \frac{1}{\triangle}\int_\Omega P(\overline{\phi})\dot{\overline{\phi}}N_j d\Omega, \tag{4.17}$$

for $j = 1,...,n$.

We can now express this equation system in the matrix form:

$$\frac{\varepsilon^2}{m}M\underline{\dot{\phi}} = \underline{g}(\underline{\phi},\underline{u}), \tag{4.18}$$

$$M\underline{\dot{u}} = -K\underline{u} + \underline{f}(\underline{\phi},\underline{\dot{\phi}}). \tag{4.19}$$

Here M is the usual $n \times n$ mass matrix with entries $M_{ij} = \int_\Omega N_i N_j d\Omega$ and K is the $n \times n$ stiffness matrix with entries $K_{ij} = \int_\Omega \overrightarrow{\nabla} N_i \cdot \overrightarrow{\nabla} N_j d\Omega$. The vector $\underline{g} = \underline{g}(\underline{\phi}, \underline{u})$ represents the right-hand sides of Equation (4.16) and $\underline{f} = \underline{f}(\dot{\underline{\phi}}, \underline{\phi})$ consists of the final three terms of Equation (4.17).

4.4.2 Semi-Implicit Discretisation in Time

The theta method, introduced in Section 2.3.2, may be used to achieve a semi-implicit time-stepping scheme (as in [97] for example). The simpler temperature equation system, (4.19), is discretised in an implicit manner while the more complex phase system, (4.18), is discretised in an explicit manner. The implicit discretisation will improve the overall efficiency of the time stepping scheme while the explicit part suppresses the highly non-linear terms in equation system (4.18). (A fully implicit scheme will be introduced in Chapter 7.)

In order to make it clear how the theta method is applied the matrix form of the phase-field equation is rearranged to be:

$$\dot{\underline{\phi}} = \frac{m}{\varepsilon^2} M^{-1}[\underline{g}].$$

Applying the theta method gives:

$$\frac{\underline{\phi}^{k+1} - \underline{\phi}^k}{\Delta t} = (1-\theta)\frac{m}{\varepsilon^2} M^{-1}[\underline{g}^k] + \theta \frac{m}{\varepsilon^2} M^{-1}[\underline{g}^{k+1}],$$

where $\underline{\phi}^k$ is the approximate solution resulting from time step k. It follows that $\underline{\phi}^{k+1}$ is the approximate solution after the following time step. As we wish to solve the phase equation explicitly, we select $\theta = 0$ and the equation system reduces to:

$$\frac{\underline{\phi}^{k+1} - \underline{\phi}^k}{\Delta t} = \frac{m}{\varepsilon^2} M^{-1}[\underline{g}^k].$$

Separating the $\underline{\phi}^k$ terms and the $\underline{\phi}^{k+1}$ terms gives:

$$\frac{\varepsilon^2}{m} M \underline{\phi}^{k+1} = \frac{\varepsilon}{m} M \underline{\phi}^k + \Delta t \underline{g}^k. \tag{4.20}$$

Since the $\underline{\phi}^k$ values are known, the nonlinearity of $\underline{g}$ is suppressed and this method results in a system of linear equations. The temperature system can be treated in a similar manner. The matrix form (4.19) is rearranged to:

$$\dot{\underline{u}} = M^{-1}[-K\underline{u} + \underline{f}].$$

Applying the theta method gives:

$$\frac{\underline{u}^{k+1} - \underline{u}^k}{\Delta t} = (1-\theta) M^{-1}[-K\underline{u}^k + \underline{f}^k] + \theta M^{-1}[-K\underline{u}^{k+1} + \underline{f}^{k+1}]$$

Since we wish to solve the temperature equation implicitly, we select the simplest such scheme for which $\theta = 1$, reducing the system to:

$$\frac{\underline{u}^{k+1} - \underline{u}^k}{\Delta t} = M^{-1}[-K\underline{u}^{k+1} + \underline{f}^{k+1}].$$

The resulting system is now rearranged to collect together the $\underline{u}^{k+1}$ terms. The vector $\underline{f}^{k+1}$ is not considered as a variable term as it has a known value and does not depend on $\underline{u}^{k+1}$. The ϕ^{k+1} values on which $\underline{f}^{k+1}$ does depend have been calculated in the solution of system (4.20). The final version of the fully discretised temperature equation system is therefore:

$$[M + \Delta t K]\underline{u}^{k+1} = M\underline{u}^k + \Delta t \underline{f}^{k+1}. \tag{4.21}$$

The solution of the linear algebraic systems (4.20)and (4.21) on a uniform grid is relatively straightforward: a standard iterative solver, such as the CG method, may be used. In the case of a non-uniform grid (containing hanging nodes) this is no longer the case.

4.4.3 Projected Preconditioned Conjugate Gradient

The grid adaptivity methods, developed in this Section 4.3 of this chapter, may be used in the solution of the semi-implicit PF model. As mentioned in Section 3.4.2 the inter-phase region of these models requires a level of refinement not necessary for the rest of the domain. The use of h-adaptive FE methods allows this region to be resolved at a reduced cost in comparison to global refinement methods. Since we have chosen to leave the hanging nodes in the grids unresolved, the solver used must resolve these values in order to prevent discontinuities in the solution. To achieve this we make use of the Projected Preconditioned Conjugate Gradient Method (PPCG) introduced by Meyer in [67]. The PPCG is an adaptation of the standard CG method (described in Section 2.3.2). It allows assembly of a non-conforming FE matrix by combining the element matrices without regard to whether any of the element vertices are hanging nodes. A projection step is then added to the CG algorithm to ensure that the solution values of the hanging nodes are constrained to force continuity of the solution. The principles of this modified CG method are discussed here.

We represent our non-conforming linear system of equations as:

$$\hat{A}\underline{u} = \underline{\hat{b}}, \tag{4.22}$$

where the matrix $\hat{A}$ and the vector $\underline{\hat{b}}$ both include the FE assembly of all elements regardless of whether any vertices are hanging nodes. A direct solution of this system would therefore be discontinuous and a poor representation of the true PDE solution. An example of a discontinuity along an edge containing a hanging node can be seen in Figure 4.7. The form of the solution vector is $\underline{u} = [u_0, ..., u_p, ..., u_q, ..., u_r, ...u_n]^T$. We assume that there is at least one value corresponding to a hanging node, u_q, with another two values corresponding to each of its parent nodes, u_p and u_r. A projection matrix, P, is

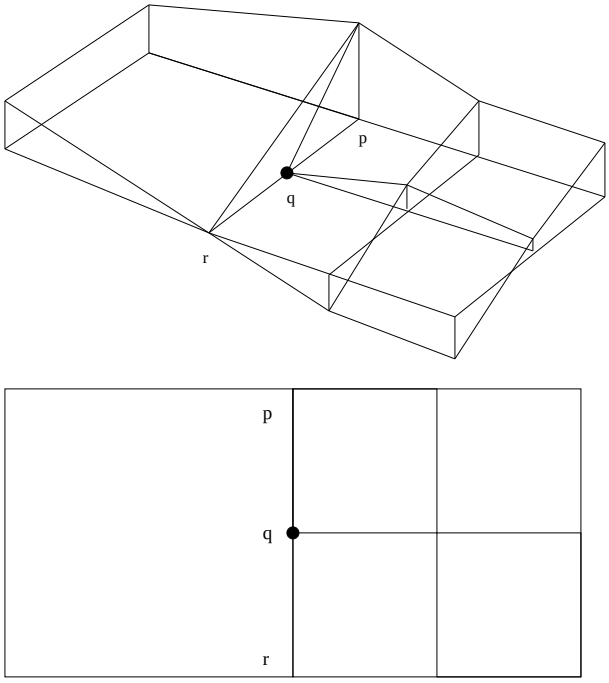


Figure 4.7: A discontinuity due to a hanging node: An example of a surface over a non-uniform, two-dimensional grid with a discontinuity along an edge containing a hanging node. The hanging node is marked with a filled circle.

introduced which has the effect of replacing the values at the hanging nodes by a linear interpolant of the values at their parent points. So, if q were the only hanging node, $P\underline{u} = [u_0, ..., u_p, ..., \frac{1}{2}(u_p + u_r), ..., u_r, ...u_n]^T$. The transpose of this matrix, P^T, is also used. The effect of this operation is: $P^T\underline{u} = [u_0, ..., u_p + \frac{1}{2}u_q, ..., 0, ..., u_r + \frac{1}{2}u_q, ...u_n]^T$. We then consider a modification of Equation (4.22) so that we actually solve the system

$$P^T\hat{A}P\underline{u} = P^T\underline{\hat{b}}. \tag{4.23}$$

Using a simple illustrative system, containing only a hanging node and its two parent nodes, we can see that the projection matrices are singular, e.g.

$$\begin{bmatrix} 1 & \frac{1}{2} & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{2} & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix},$$

where this system corresponds to System (4.23). However, the two equations resulting from this system,

$$(a_{11}+\frac{1}{2}a_{21})u_1+(a_{12}+\frac{1}{2}a_{22})\frac{1}{2}(u_1+u_3)+(a_{13}+\frac{1}{2}a_{23})u_3=b_1+\frac{1}{2}b_2$$

$$(a_{31}+\frac{1}{2}a_{21})u_1+(a_{32}+\frac{1}{2}a_{22})\frac{1}{2}(u_1+u_3)+(a_{33}+\frac{1}{2}a_{23})u_3=b_3+\frac{1}{2}b_2$$

contain only two variables. In general, we effectively solve a $(n-hn)\times(n-hn)$ system, where n is the total number of points and hn is the number of hanging nodes. The hanging node values are maintained through the restriction formulae, e.g. $u_2=\frac{1}{2}(u_1+u_3)$.

We do not wish to explicitly form (4.23) however: instead we seek to use our assembled system (4.22). The use of the projection matrices allows us to define two vector spaces: a conforming space, $\underline{U}$ (for which the hanging node values are always the average of their parents and so the corresponding FE functions are always continuous) and the nonconforming space $\underline{V}$ (for which all the nodes are free and functions may be discontinuous along edges containing hanging nodes). For a given vector, $\underline{v}\in R^n$: $P\underline{v}\in\underline{U}$ and $P^T\underline{v}\in\underline{V}$. The conjugate gradient search is performed in the nonconforming space while the solution is maintained in the conforming one. The algorithm for the PPCG method is given in Algorithm 8. Here, $\underline{u}$ represents the approximation to the solution, $\underline{r}$ and $\underline{w}$ are both forms of the residual and $\underline{q}$ is the search direction. In addition a preconditioner is introduced: $C^{-1}:\underline{V}\rightarrow\underline{U}$. In the simplest case $C^{-1}=PP^T$. In this implementation a diagonal preconditioner, D^{-1}, was used, so $C^{-1}=PD^{-1}P^T$.

4.4.4 Implementation

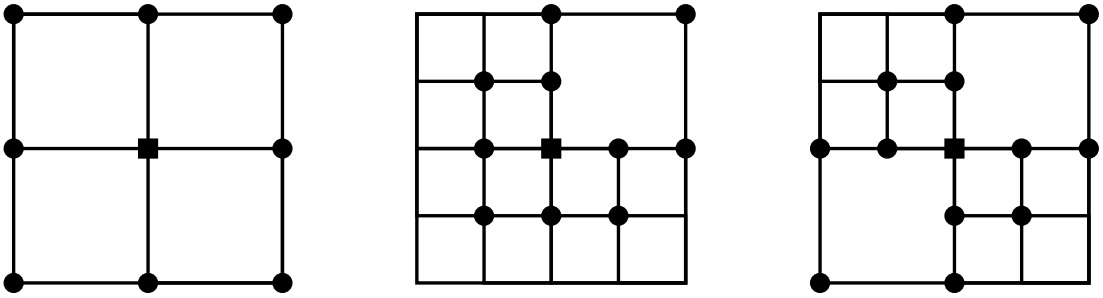
A selection of the issues associated with the implementation of this solution process are discussed in this section.

Algorithm 8 The Projected Preconditioned Conjugate Gradient algorithm. (Meyer [67])

$\underline{u} \in U$
$\underline{r} = \hat{A}\underline{u} - \hat{\underline{b}}$
$\underline{q} = \underline{w} = C^{-1}P^T\underline{r}$
$\gamma = (\underline{w}, \underline{r})$
while Stopping criterion is not met **do**
  $\delta = (\underline{q}, \hat{A}\underline{q})$
  $\alpha = \frac{-\gamma}{\delta}$
  $\tilde{\underline{u}} = \underline{u} + \alpha\underline{q}$
  $\tilde{\underline{r}} = \underline{r} + \alpha\hat{A}\underline{q}$
  $\tilde{\underline{w}} = C^{-1}P^T\tilde{\underline{r}}$
  $\tilde{\gamma} = (\tilde{\underline{w}}, \tilde{\underline{r}})$
  $\beta = \frac{\tilde{\gamma}}{\gamma}$
  $\tilde{\underline{q}} = \tilde{\underline{w}} + \beta\underline{q}$
end while

Matrix assembly

The matrices that are calculated for the solution of the PF problem are $n \times n$ as we assemble for all of the internal points in the grid, and in order to produce an accurate solution we will use numbers of internal nodes of the order of $\mathscr{O}(10^6)$ (in the uniform case). To avoid storing a 2D array of this size a sparse matrix structure is required. This data structure exploits the local nature of the FE method to restrict the number of values stored. The definition of the FE basis functions dictates that an entry i, j in the matrix is only non-zero if point i is joined to point j in the mesh through at most one edge. If we were using globally refined grids there would therefore be nine entries on each row of the matrix. However, with our locally refined grid there can be up to thirteen entries on a matrix row, which corresponds to one internal node. Three of the possible refinement arrangements, around a point, are shown in Figure 4.8. The matrices are formed as in a conventional assembly procedure, with a loop over the elements. For example, to build the mass matrix, M, we follow the steps in Algorithm 9. Note that for simplicity of exposition this Algorithm uses a two-dimensional array $M[i][j]$ rather than the sparse data structures used in practice. A sparse matrix data structure is presented in [39] where a set of linked lists



9 Entries 11 Entries 13 Entries

Figure 4.8: Three of the possible refinement configurations around a non-uniform grid point: resulting in a number of entries in the sparse matrix row corresponding to that point. The number of entries is stated below each grid fraction.

Algorithm 9 Assembly of a standard mass matrix over a grid of finite elements

```
1:  j = i = 0
2:  e = 0
3:  while j ≤ n do
4:     while i ≤ n do
5:        M[j][i] = 0.0
6:        i++
7:     end while
8:     j++
9:  end while
10: while e ≤ number of elements do
11:    j = i = 0
12:    while j ≤ 3 do
13:       J = global index of node j of element e
14:       if J is an internal point then
15:          while i ≤ 3 do
16:             I = global index of node i of element e
17:             if I is an internal point then
18:                M[j][i] += ∫_{Ω_e} N_J N_I dΩ
19:             end if
20:             i++
21:          end while
22:       end if
23:       j++
24:    end while
25:    e++
26: end while
```

Number of safety layers	0	1	2	3
Number of steps before re-adaption	5	10	20	40

Table 4.1: Number of safety layers and corresponding re-adaption points: The number of time steps that can be taken before grid re-adaption is required for various numbers of safety layers of refinement, are given.

are used, each representing a column of the matrix. The entries are stored in order and zero entries are omitted from the lists. Since it is possible to establish from the nature of our gridding scheme the maximum number of entries on a row of the matrix, we store a $13 \times n$ array of 'matrix entry' structures. This array represents the n rows of the matrix each with a potential of 13 entries. Each structure contains its solution value as well as its column number. The diagonal entries of the matrix, which for this FE scheme are always non-zero, are stored in the first entry of each 'row' to facilitate diagonal preconditioning. A set of simple complementary functions were written for the insertion of matrix entries and the access to matrix values.

Frequency of Grid Re-Adaption

Increasing the number of 'safety layers' of refinement that we place around the refinement created as a result of the error estimator helps to guarantee correct resolution for longer and allows us to re-adapt them less frequently. However, adding layers of refinement increases the size of the algebraic system and the expense of the solution process. We therefore need to find a balance between these two things. The choice of compromise between the two is clearly dependent on the application and on the size of the time step taken. For the results in this chapter the refinement tolerance ($rtol$) was set to 10^{-3} and the coarsening tolerance was set in line with this. Qualitative tests were carried out to discover how many steps could be taken before re-refinement was required for various numbers of safety layers. Table 4.1 gives the number of time steps that can be taken before grid re-adaption for various numbers of safety layers. The results in Table 4.1 apply for grid level 7 to level 11 and the time step sizes which were used for each level are

Maximum Grid Level	7	8	9	10	11
Max Δt	2.0e-04	8.0e-05	2.0e-05	5.0e-06	1.0e-06

Table 4.2: Semi-implicit PF model: Maximum stable time step: The maximum stable time step size is shown for various maximum grid refinement levels on a domain $\Omega = (-1,1) \times (-1,1)$.

detailed in Section 4.4.5.

Time profiling on this implementation (more details of which are given in Section 4.4.5) shows that the solution process is by far the most expensive operation. We are therefore inclined towards adapting the grid frequently and minimising the system size. The results in this chapter (including the time profiling results) were therefore found with the cautious choice of 2 safety layers of refinement and re-adaption every other step.

Time Stepping with a Semi-Implicit Discretisation Scheme

As the phase equation has been discretised in an explicit manner we would expect there to be a restriction on the maximum stable time step size, Δt. The maximum time step size is linked to the size of the finest grid spacing in the mesh. Trial and error tests were run to establish the maximum time step for a range of maximum refinement levels. These are given to one significant figure in Table 4.2. Stability analysis for linear parabolic problems (e.g. [86]) implies that we would expect that halving the minimum grid spacing would mean a reduction of a factor of four in the maximum stable time step size. The results shown in Table 4.2 correlate well with this assessment.

4.4.5 Results

Evidence of the performance of our adaptive meshing scheme applied to a PPCG solution of a PF problem is given in this section. Results are given for the grids with at least level 7 refinement. Grids of coarser refinement than this display significant overshoot around the solid-liquid interface and the interface is not captured sufficiently well for any interesting

Grid Level	No. Time Steps	$\lvert\phi_{glo} - \phi_{loc}\rvert_{L_2}$	$\lvert u_{glo} - u_{loc}\rvert_{L_2}$
7	5	0.0000056132	0.0000620440
8	13	0.0000008498	0.0000935515
9	50	0.0000081953	0.0000802204

Table 4.3: Semi-implicit PF model: Global and local refinement solution comparison: The L_2-norm of the difference between solutions found on globally and locally refined grids, taken at $t = 0.001$.

solutions to be observed.

Comparison of Locally and Globally Refined Meshes

In order to verify the accuracy of the adaptive meshing technique we compare the solutions produced with a locally refined grid (ϕ_{loc} and u_{loc}) with those produced with a globally refined grid (ϕ_{glo} and u_{glo}). It is necessary to use a relatively coarse mesh for this test in order to have a practical execution time for the global refinement case. The maximum refinement level in the locally refined grid corresponds to the refinement level everywhere in the globally refined grid. The locally refined grids are created using the methods described in this chapter and the refinement and coarsening are driven by an error indicator based on the gradient of the ϕ variable. As is shown later in this section the temperature variable, u, changes much more gradually than the phase variable which has a very small area of rapid change in the interface region. Elsewhere in the domain the phase field variable has an insignificant gradient. Therefore when a refinement pass is made over the grid, any element whose gradient exceeds 10^{-2} is further refined. The difference between solutions in the two refinement cases (after a fixed time period $t = 0.001$) are given in Table 4.3, for three choices of maximum refinement level. (Details of how these differences are found are given in Section 6.3.3.) There is little difference between the discrepancies from one grid level to the next at this stage of the simulation. The way that the difference between the globally and locally refined solutions changes as solutions are produced for further time steps is shown in Figure 4.9. The error is small in comparison to the scale of changes in the solution (which are of order 1) however it seems to

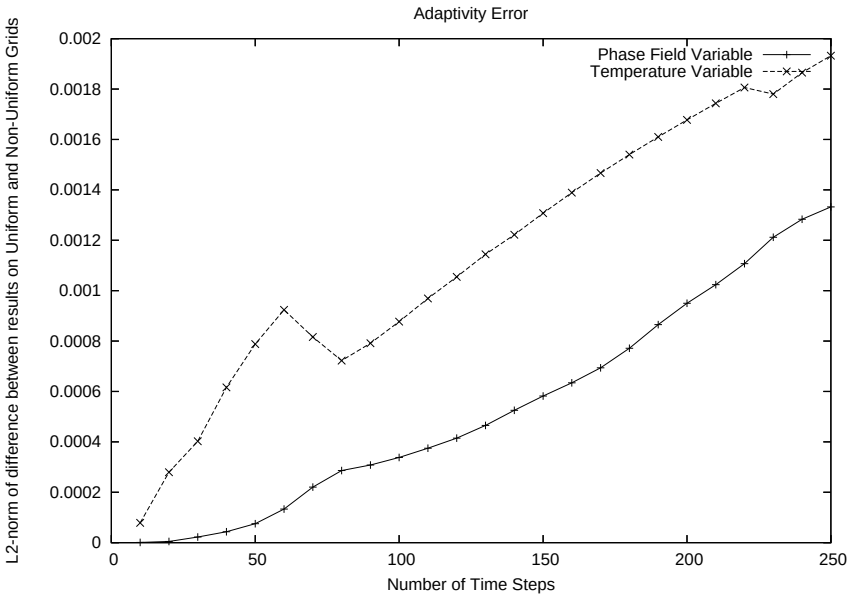


Figure 4.9: Semi-implicit PF model: Adaptivity error with time: The growth of adaptivity error as more time steps are taken is shown. These results are for a maximum refinement level of 8.

increase almost linearly as more time steps are taken: this suggests a very small (approximately constant) error at each time step, as one would expect.

Table 4.4 compares the number of nodes and total execution time for the globally refined and locally refined cases. In the case of the globally refined grid the execution time per step will remain roughly the same throughout the execution. However, in the locally refined case the number of nodes may change from one time step to the next and therefore so will the execution time per step. For this particular application the number of nodes in the locally refined grid will steadily increase throughout the solution process as the interface region of the phase variable lengthens. There is a difference of at least an order of magnitude in the execution time between the globally and locally refined cases for every choice of maximum grid level shown. As the maximum grid level increases the difference in execution time between the two cases naturally increases.

Figure 4.10 shows the number of grid points plotted against the execution time per step

	Global Refinement		Local Refinement	
Grid level	No. Nodes	Exec Time (secs)	No. Nodes	Exec Time (secs)
7	16129	50	717	5
8	65025	625	1433	15
9	261121	10000	3537	127
10	1046529	158000	9285	1266

Table 4.4: Semi-implicit PF model: System size and execution time: The number of nodes and total execution time for various grid levels are given. All solutions where calculated up to $t = 0.001$. (The finer the grid refinement then the more time steps are required to reach this time due to the restriction on the maximum step size.)

throughout the adaptive solution process. As this relationship is roughly proportional we conclude that there are no significant, hidden computational expenses in the adaptive gridding scheme and that the cost per step is proportional to the number of unknowns for a given maximum refinement level. The linearity of solution time against system size comes about because the mass matrix, M, is very well conditioned meaning that the number of PPCG iterations is more or less independent of the system size for (4.20) and also for (4.21) (provided that Δt is small).

Performance of Solution Method

The convergence rate of the PPCG in this context is shown in Figure 4.11. There is a relatively small difference between the convergence of solutions produced on grids of different maximum refinement levels. On average the residual inner product is reduced by a factor of approximately three at each iteration. As mentioned above both of the systems solved for this problem are dominated by a Galerkin mass matrix (provided that Δt is small). The fast convergence that is shown in Figure 4.11 results from the conditioning properties of this mass matrix, documented in [96].

We now look at the reduction in the error as the maximum grid level is increased. As the FE method is second order we would expect this error to reduce by roughly a factor of four each time the maximum grid level is increase by one level. Since we have no exact

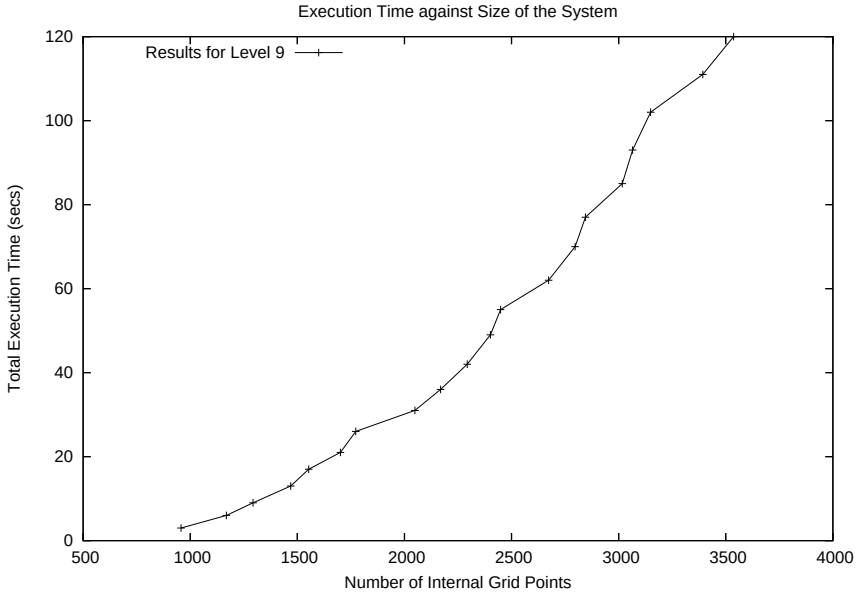


Figure 4.10: Semi-implicit PF model: Execution time against the size of the system: The relationship between the number of nodes and the execution time is shown for a locally refined solution with a maximum grid level of 9.

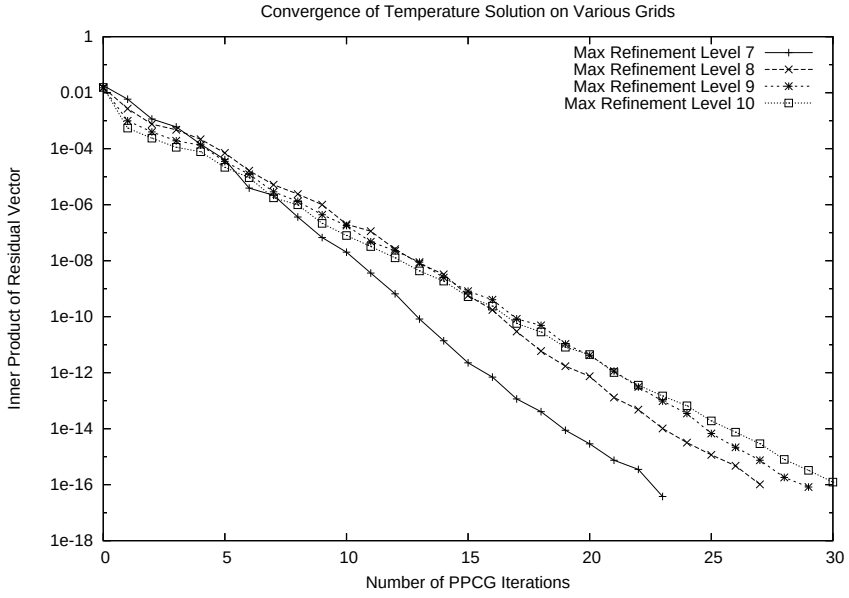


Figure 4.11: Semi-implicit PF model: Convergence of the PPCG: The inner product of the residual vector is plotted against the number of iterations of the PPCG. The residual inner products are shown on a logarithmic scale for various locally refined grids.

Grid level	$\|\phi^{\ell} - \phi^{\ell+1}\|_{L_2}$	$\|u^{\ell} - u^{\ell+1}\|_{L_2}$
$\ell = 6$	0.0283277936	0.0150150278
$\ell = 7$	0.0029681114	0.0011015478
$\ell = 8$	0.0006905591	0.0005329332
$\ell = 9$	0.0001951886	0.0001620033
$\ell = 10$	0.0000593247	0.0000442036

Table 4.5: Semi-Implicit PF model: Spatial error convergence: L_2-norm of difference between solutions from various grids. For example, the line $\ell = 7$ shows the L_2-norm of the difference between the solutions produced with a maximum grid refinement of level 7 and those produced with a maximum grid refinement of level 8.

solution for this problem we look at the difference between the solutions at various maximum grid levels. In order to compare the solutions from two grids of different maximum level both solutions are projected onto a uniform grid using linear interpolation where necessary. Calculating the difference between the two is then trivial, with the L_2-norm of this difference vector. The norm is calculated using a simple quadrature rule:

$$\sum_{e=1}^{ne} \frac{h^2}{4}(u_1^2 + u_2^2 + u_3^2 + u_4^2),$$

where *ne* is the number of elements in the grid, h is the size of the element and u_1 to u_4 are the four solution values associated with that element. In Table 4.5 we can see that the L_2 norm of the difference between the solutions reduces by a factor of approximately four each time the grids become finer.

Time Profiling of Implementation

A profile of the percentage of execution time that was spent in each of the three main tasks: FE assembly, adaptive meshing and system solution is given in Table 4.6 for three grid levels. Each level was monitored for 500 time steps. It is clear the solution process takes the vast majority of the execution time in all of these cases. The meshing algorithms however take the least time of these three main tasks. The adaptive meshing therefore

Level	PPCG Solver	FE Assembly	Adaptive Meshing
7	67.8	28.2	2.6
8	78.6	18.4	2.1
9	86.9	11.1	1.4

Table 4.6: Semi-implicit PF model: Time profiling for a semi-implicit implementation of a phase-field model using adaptive meshing and a modified Conjugate Gradient solver. The figures show the percentage of the total execution time, taken for each type of process.

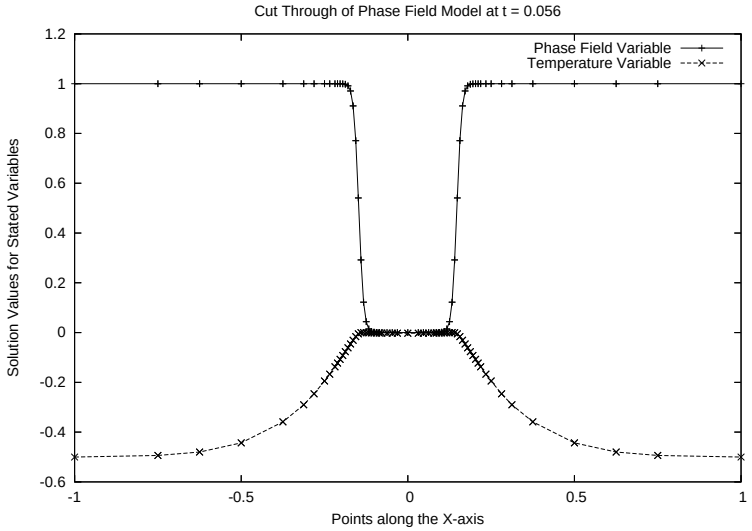


Figure 4.12: Semi-implicit PF model: The phase field and temperature approximations after 700 time steps on a grid with a maximum refinement level of 8. Solutions are only shown for points along the x-axis. These points are naturally unevenly spread as the grid becomes more refined closer to the interface.

allows for many fewer unknowns in the algebraic system without adding any significant overhead to the execution time.

Visualisation of Results and Meshes

Figure 4.12 presents an outline of both solution variables along the line $y = 0$. We can see here the difference in scale over which the phase variable changes in comparison to the temperature variable. The phase variable solutions for a longer run is shown in Figure 4.13. The three pictures are from time steps 500, 1300 and 2200 respectively. With this semi-implicit scheme many time steps are required to produce the solution of interest.

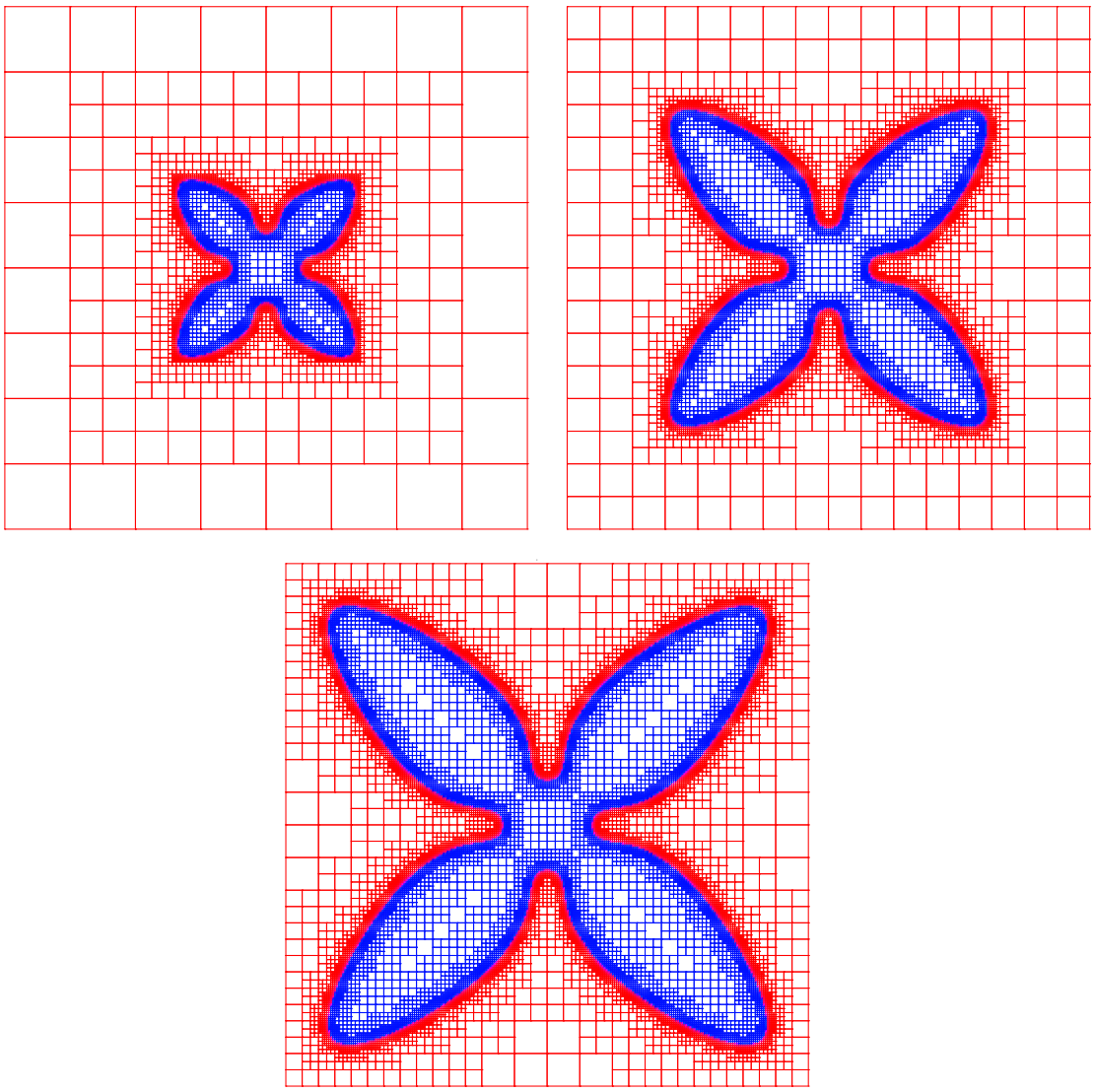


Figure 4.13: Semi-implicit PF model: Two-dimensional solutions and grids: The adaptively refined grid for the PF simulation at $t = 0.4$, $t = 1.04$ and $t = 1.76$. The blue parts of the grid represent areas where $\phi < 0.5$ which in a physical sense represents the solid part of the material. The red parts of the grid represent $\phi > 0.5$ and the liquid part of the material. (Coarser areas of the grid, surrounding those that are shown, have been cropped.) These results were produced with a maximum refinement level of 8.

Chapter 5

Projected Multigrid Method

5.1 Introduction

In this chapter we present the main original contribution of this thesis which is based upon combining the projection algorithm of Meyer [67] that was introduced in Section 4.4.3 with the FAS multigrid scheme described in Section 2.4.2. We have chosen to develop a method for the solution of nonlinear elliptic and parabolic problems over non-uniform grids containing hanging nodes. This allows the use of bilinear quadrilateral finite elements for adaptive meshing without the overhead associated with extra triangular refinement for the resolution of the hanging nodes. Our method is derived from an existing nonlinear, iterative solution method and enriched with the capability to resolve the values at the hanging nodes during the solution process. We are interested in solving time-dependent, parabolic problems which produce a nonlinear algebraic system at each time step. The efficiency of the algebraic solver is therefore important, especially as it has been shown (in Section 4.4.5) to be the most time consuming procedure of the implementation. A multigrid method is used as the outer iteration of our solution method as these have

been shown to provide significantly accelerated convergence, as described in Section 2.4.

Our method uses the standard FAS multigrid scheme, along with a modified Jacobi smoother to appropriately deal with the hanging nodes in locally refined grids. The smoother is based on a nonlinear Jacobi iteration, presented in Section 5.2. The projection method introduced by Meyer [67] for the CG method, is applied to the nonlinear Jacobi smoother to form a new Projected Jacobi iteration. Details of the Projected Jacobi method are given in Section 5.3. This allows multigrid corrections to the solution to be calculated in a sequence of non-conforming spaces, where contributions from the hanging nodes are assembled in the normal manner in the FE method. These corrections are projected into the conforming space (for which the hanging nodes' values are linear interpolants of adjacent node values) before being added to the solution. In Section 5.5, various details of the FAS multigrid method are therefore modified to ensure that it preserves the necessary vector spaces for the Projected Jacobi smoother. This version of the FAS method is referred to as Projected FAS (PFAS) multigrid. The definition of a grid level used for the traditional version of the FAS, in Algorithm 2 (in Section 2.4.2), does not apply to the locally refined grids that the Projected FAS is designed for. A method for defining grid levels in a locally refined grid is given in Section 5.4. For the simplicity of this introduction the new method is described for the solution of a nonlinear elliptic model problem and the performance for this problem is given in Section 5.6. However the generalisation to the solution of parabolic equations and systems, using implicit time stepping, is presented in the following chapter.

5.2 Nonlinear Gauss-Seidel Smoother

The Jacobi and Gauss-Seidel (G-S) iterative solvers were introduced in Section 2.3.2 for the solution of linear systems of equations. Here we look at the derivation of a nonlinear

G-S and, in Section 5.3, show how this method is modified to create a Jacobi smoother for the solution of systems defined over locally refined grids with hanging nodes. The following formulation of the nonlinear G-S method is similar to that in [19, pp 95-98]. We represent a general nonlinear system of equations derived from the discretisation of a nonlinear elliptic PDE as

$$\underline{K}(\underline{u}) = \underline{b}, \tag{5.1}$$

where $\underline{K}$ is a nonlinear function, $\underline{b}$ is a known vector and $\underline{u}$ is the unknown solution vector. In one step of a G-S sweep, one value in the solution vector is updated using the current approximation. This is mathematically equivalent to setting the corresponding value in the residual vector to zero. Therefore, the operation is calculating a correction to the existing solution, which can be represented as

$$\underline{K}(\underline{u} + \delta_j \underline{\varepsilon}_j) = \underline{b}, \tag{5.2}$$

where $\underline{\varepsilon}_j$ is the j^{th} unit vector and δ_j is the correction for solution value j. Since we are setting the j^{th} residual entry to zero we can write, $[\underline{K}(\underline{u} + \delta_j \underline{\varepsilon}_j)]_j - b_j = \underline{0}$. Expanding the function, $\underline{K}$, using Taylor series gives:

$$\underline{K}(\underline{u} + \delta_j \underline{\varepsilon}_j) \approx \underline{K}(\underline{u}) + \delta_j \frac{\partial \underline{K}(\underline{u})}{\partial \underline{u}} \underline{\varepsilon}_j + \mathscr{O}(\delta_j^2).$$

Discarding the $\mathscr{O}(\delta_j^2)$ terms and using Equation (5.2) we can rearrange this equation to be:

$$[\underline{K}(\underline{u}) + \delta_j \frac{\partial \underline{K}(\underline{u})}{\partial \underline{u}} \underline{\varepsilon}_j]_j - b_j \approx 0.$$

Now we can rearrange this equation to get the expression for δ_j:

$$\delta_j \approx \frac{(b_j - [\underline{K}(\underline{u})]_j)}{[\frac{\partial \underline{K}(\underline{u})}{\partial \underline{u}} \underline{\varepsilon}_j]_j}. \tag{5.3}$$

This is a scalar equation resulting in a value for the correction δ_j which is then added to the appropriate entry of the solution ($u[j] = u[j] + \delta_j$) before moving to the next j. There are three terms that must be calculated for this approximation. The source term, b_j, is simply the j^{th} entry of the source term vector, $\underline{b}$. This vector need only be calculated once per G-S solve. The function, $\underline{K}(\underline{u})$ results in a vector and $[K(\underline{u})]_j$ is the j^{th} entry of that vector. For a true G-S method this vector should be re-calculated after each correction. The Jacobian term is a little more complicated. The differential component, $\frac{\partial \underline{K}}{\partial \underline{u}}(\underline{u})$, defines the Jacobian matrix:

$$\begin{bmatrix} \frac{\partial K_1}{\partial u_1}(\underline{u}) & \frac{\partial K_1}{\partial u_2}(\underline{u}) & \dots & \frac{\partial K_1}{\partial u_n}(\underline{u}) \\ \frac{\partial K_2}{\partial u_1}(\underline{u}) & \frac{\partial K_2}{\partial u_2}(\underline{u}) & \dots & \frac{\partial K_2}{\partial u_n}(\underline{u}) \\ \dots & \dots & \dots & \dots \\ \frac{\partial K_n}{\partial u_1}(\underline{u}) & \frac{\partial K_n}{\partial u_2}(\underline{u}) & \dots & \frac{\partial K_n}{\partial u_n}(\underline{u}) \end{bmatrix}.$$

Since the only non-zero entry of $\underline{\varepsilon}_j$ is entry j (which is 1) we may write $[\frac{\partial \underline{K}}{\partial \underline{u}}\underline{\varepsilon}_j]_j$, the denominator of (5.3), as:

$$\frac{\partial K_j(\underline{u})}{\partial u_j}.$$

Given these three components, a correction can be calculated in turn for each entry in the solution vector. This represents a single G-S iteration. A Jacobi version of the smoother would perform the same operations but simply postpone the update of the solution until all of the corrections $j = 1, \dots, n$ had been calculated.

5.3 Nonlinear Projected Jacobi Smoother

In this section the smoother defined in the Section 5.2, is modified to form a Jacobi iteration that is effective on a system of nonlinear equations obtained from a finite element discretisation on a locally refined grid with hanging nodes. We are attempting to solve a nonlinear system

$$\hat{\underline{K}}(\underline{u}) = \hat{\underline{b}}, \tag{5.4}$$

where $\underline{\hat{K}}(\underline{u})$ is the vector obtained from the assembly of the element stiffness matrices without regard for the hanging nodes. That is the hanging nodes receive contributions from the surrounding elements, in the FE method, but their element support area is incomplete and therefore so is their FE assembly. Taking from the theory of Meyer's PPCG method [67] (described in detail in Section 4.4.3) we actually solve the system:

$$P^T \underline{\hat{K}}(P\underline{u}) = P^T \underline{\hat{b}}, \tag{5.5}$$

where P is the matrix which projects a vector in R^n (where n is the number of non-Dirichlet nodes) into the conforming space. The effect of P is to overwrite the value of each non-Dirichlet hanging node value with a linear interpolation of its parent point values. The operator P^T is the transpose of P. These two operators relate to the two spaces which contain the vectors which are significant for this method. The first is a conforming space which we denote as $Image(P)$. The second vector space is $Image(P^T)$, the non-conforming space. It is important to note that, given $\underline{u} \in R^n$, $P^T P\underline{u} \neq PP^T\underline{u} \neq \underline{u}$. The action of the projection operators on the vectors involved in the iteration solution process, significantly complicates the nonlinear Gauss-Seidel update of the solution. The smoothing method is therefore adjusted so that a Jacobi update is employed. This update, δ_j, is calculated for $j = 1,...,n$. A vector, $\underline{\delta} = [\delta_1,...,\delta_n]^T$ can then be constructed and the correction is performed at the end of the sweep:

$$\underline{u}^{(r+1)} = \underline{u}^{(r)} + \underline{\delta},$$

(where $\underline{u}^{(r+1)}$ is the new approximation of the solution). Applying a standard nonlinear Jacobi iteration, to this new projected system yields a correction vector, $\underline{\delta}$, that is defined point-wise as:

$$\delta_j = \mu \left[\frac{[P^T \underline{\hat{b}} - P^T \underline{\hat{K}}(P\underline{u})]_j}{[\frac{\partial}{\partial u_j} P^T \underline{\hat{K}}(P\underline{u})]_j} \right], \tag{5.6}$$

for $j = 1,...,n$ (where μ is an under-relaxation parameter). The resulting vector $\underline{\delta}$ is not generally in the conforming space and so must be projected into the conforming space before being used to augment the solution. For clarity the correction vector (5.6) for this method can be simplified to

$$\delta_j = \mu \left[\frac{d_j}{e_j} \right], \tag{5.7}$$

where $j = 1,...,n$, and d_j and e_j are components of the following vectors $\underline{d}$ and $\underline{e}$ respectively:

$$\underline{d} = P^T \underline{\hat{b}} - P^T \underline{\hat{K}}(P\underline{u})$$

and

$$\underline{e} = \frac{\partial}{\partial u_j} P^T \underline{\hat{K}}(P\underline{u}).$$

Currently, both $\underline{d}$ and $\underline{e}$ are contained in the non-conforming vector space. However, if we project these vectors to

$$\underline{d}' = P[P^T \underline{\hat{b}} - P^T \underline{\hat{K}}(P\underline{u})]$$

and

$$\underline{e}' = P[\frac{\partial}{\partial u_j} P^T \underline{\hat{K}}(P\underline{u})],$$

they are both contained in the conforming space. A valid component-wise Jacobi correction would then be:

$$u_j^{(r+1)} = u_j^{(r)} + \mu \left[\frac{d'_j}{e'_j} \right]. \tag{5.8}$$

Note that, although this correction is defined in a point-wise manner, all components of the correction vector are calculated before the solution is augmented (i.e. a Jacobi iteration is used).

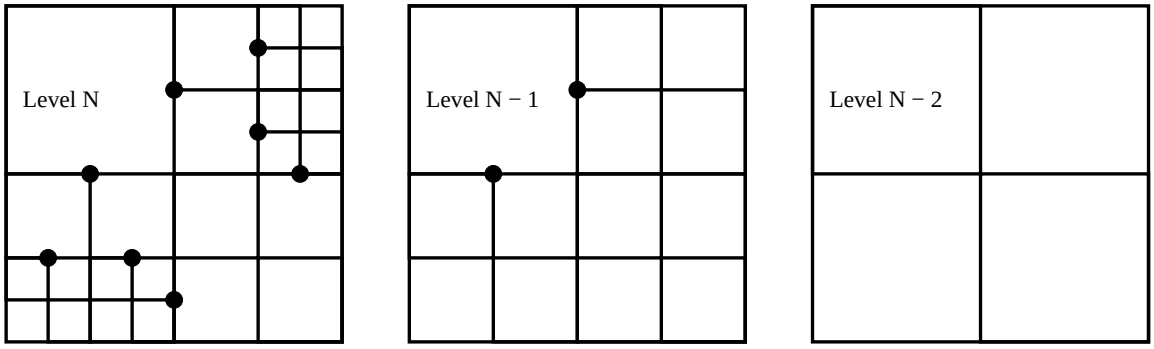


Figure 5.1: Multigrid composite grid levels: Three non-uniform composite grid levels are shown including hanging nodes marked by filled circles.

5.4 Composite Grids for the Full Approximation Scheme

The h-adaptive gridding scheme presented in Chapter 4 is used again here. In order to use the FAS multigrid method on such grids, what is meant by a level in the grid must be defined with respect to the new FAS algorithm. To achieve this a new set of levels are imposed on the grid which we refer to as "composite grid levels". The composite grid level ℓ includes all of the elements on grid level ℓ and, in the areas of the domain where elements of level ℓ don't exist, the current leaf elements are included in their place. Using this definition we create a set of composite grid levels for use with the multigrid scheme. A set of composite grid levels can be seen in Figure 5.1. Defining the composite grid levels like this means that each of the composite levels covers the whole domain and is, potentially, a locally refined grid itself. Other possibilities for the selection of grid levels from locally refined meshes are discussed in Chapter 8.

5.4.1 Traversing a Composite Grid Level

Traversal techniques must be established in order to access each composite grid level. The option of establishing linked lists for each level would prove problematic in this case. Large sections of the composite grids are shared between composite grid levels and therefore the linked lists would overlap for the affected elements. More storage

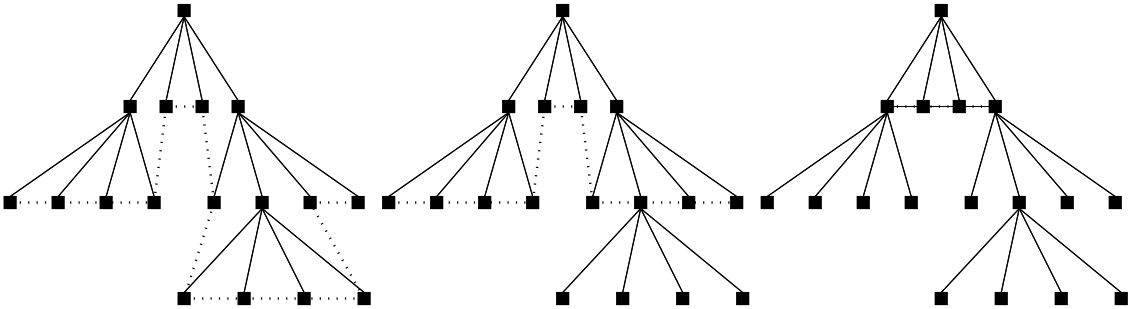


Figure 5.2: Multigrid composite level linked lists: The linked lists for three composite grid levels are shown within the quadtree data structure. While traversing a level of this tree structure using a linked list there are certain elements which have more than one option for the next element in the list as they are included on more than one grid level. Extra storage would be required to distinguish between the levels at these elements.

would be needed in order to distinguish between the composite levels for the critical links. Figure 5.2 shows a simple example of this. To avoid these linking problems a traversal algorithm was used, similar to the system for traversing the leaf elements of the quadtree. The composite grid level traversal moves through the leaf elements of the tree unless those nodes are beyond a specified level. In this case it accesses the element of the specified level. This can be implemented with a simple modification to Algorithm 3 (in Section 4.2.3). The modified version is given in Algorithm 10.

5.5 The Projected FAS Multigrid

In order to introduce the Projected FAS (PFAS) scheme we will describe a two-grid scheme. The algorithm will later be extended to a multilevel version. A locally refined grid is used and stored in a hierarchical quadtree data structure. The two grids that will be used in this PFAS scheme are:

- Composite grid N, corresponding to the leaf elements of the quadtree, which is a locally refined grid.
- Composite grid $N-1$, which may also be a locally refined grid.

Algorithm 10 Function: Goto Next Leaf Element

```
1:  Input Parameter: C - a pointer to the current element
2:  Input Parameter: cl - level of the composite grid being traversed
3:  if C != Root Element of Tree then
4:     Set V = C
5:     Set C = C.parent
6:     if  V == C.child[0] then
7:        C = C.child[1]
8:        while (C is not a leaf element) AND (C.ID[0] ≤ cl ) do
9:           C = C.child[0]
10:       end while
11:    else if V == C.child[1] then
12:       C = C.child[2]
13:       while (C is not a leaf element) AND (C.ID[0] ≤ cl ) do
14:          C = C.child[0]
15:       end while
16:    else if V == C.child[2] then
17:       C = C.child[3]
18:       while (C is not a leaf element) AND (C.ID[0] ≤ cl ) do
19:          C = C.child[0]
20:       end while
21:    else
22:       Call 'Goto Next Leaf Element' with C and cl
23:    end if
24: else
25:    Indicate that the last leaf element has been reached.
26: end if
```

The nonlinear Projected Jacobi smoother from Section 5.3 can be incorporated into the two-grid FAS algorithm in order to deal with the hanging nodes in the two composite grids in a natural and efficient manner. However, we must modify the FAS algorithm slightly to ensure that the vectors of the Jacobi solver remain within the necessary subspaces. The two-grid PFAS scheme is shown in Algorithm 11. The two-grid scheme is then extended to a multigrid scheme in Algorithm 12.

5.5.1 Two-Grid PFAS Scheme

In the two-grid scheme that we are describing, the level ℓ, in Algorithm 11, corresponds to grid N in the example above. Clearly the projection matrices, P and P^T will be of different sizes and forms for each of the grid levels. In Algorithm 11 this difference is denoted by a subscript ℓ or L. The initial solution guess, mentioned on *line 3*, is chosen to be within the conforming space. This is done by either choosing an initial solution that is clearly within the conforming space, for example $\underline{v} = \underline{0}$, or by calculating a guess and then projecting it into the conforming space using P. The vector $\underline{\hat{b}}$ is calculated and projected into the non-conforming space using $Image(P^T)$. The smoothing operations are performed over the system $P^T\hat{K}(\underline{u}) = \underline{b}$ which is equivalent to System (5.5). The Projected nonlinear Jacobi smoother employs the correction in Equation (5.8). The resulting solution approximation therefore remains within the conforming space. Since both $\underline{b}^\ell$ and $P_\ell^T\underline{\hat{K}}^\ell(\underline{v}^\ell)$ are in the non-conforming space ($Image(P^T)$), the residual vector $\underline{r^\ell}$ is therefore also contained in the non-conforming space.

It is important that the restriction operators create coarse grid vectors that are in the equivalent vector space to their fine grid counterparts. The restriction operators, I_ℓ^L and $\tilde{I}_\ell^L$, denote simple injection (whilst full weighting is used for the prolongation operator, I_L^ℓ). The restriction, I_ℓ^L, is a point-wise operator for the solution while $\tilde{I}_\ell^L$ resolves the difference in the element support area for the residual between the two grid levels. These opera-

Algorithm 11 Projected FAS Two Grid Scheme

1: ℓ = level number of current composite grid
2: $L = \ell - 1$ (level number of composite grid one level coarser than ℓ)
3: Choose initial fine grid solution $\underline{v}^\ell$ so that $\underline{v}^\ell \in Image(P)$
4: Calculate the right-hand side $\hat{\underline{b}}^\ell$ and set $\underline{b}^\ell = P_\ell^T \hat{\underline{b}}^\ell$
5: Update $\underline{v}^\ell$ by smoothing ρ_1 times with the system $P_\ell^T \underline{\hat{K}}^\ell(\underline{u}^\ell) = \underline{b}^\ell$ using Projected Jacobi
6: Find residual $\underline{r}^\ell := \underline{b}^\ell - P_\ell^T \hat{K}^\ell(\underline{v}^\ell)$
7: Restrict residual $\underline{r}^L := \tilde{I}_\ell^L(\underline{r}^\ell)$
8: Restrict solution $\underline{v}^L := I_\ell^L(\underline{v}^\ell)$
9: Calculate coarse grid right-hand side $\underline{b}^L := \underline{r}^L + P_L^T \underline{\hat{K}}^L(\underline{v}^L)$
10: Save the initial coarse grid solution $\underline{v}^L_{init} = \underline{v}^L$
11: Solve the system $P_L^T \underline{\hat{K}}^L(\underline{u}^L) = \underline{b}^L$ exactly, with the initial guess $\underline{v}^L_{init}$
12: Calculate the coarse grid correction $\underline{e}^L = \underline{v}^L - \underline{v}^L_{init}$
13: Prolongate the correction $\underline{e}^\ell := I_L^\ell(\underline{e}^L)$
14: Correct the fine grid solution $\underline{v}^\ell = \underline{v}^\ell + \underline{e}^\ell$
15: Update $\underline{v}^\ell$ by smoothing ρ_2 times with the system $P_\ell^T \underline{\hat{K}}^\ell(\underline{u}^\ell) = \underline{b}^\ell$ using Projected Jacobi

tors do not interfere with the projection method. The fine grid residual $\underline{r}^\ell$ is contained in $Image(P_\ell^T)$ before it is restricted, therefore all values corresponding to hanging nodes are zero. Therefore no information is lost at the boundaries between levels by using injection for the residual restriction. Injection is also appropriate for the solution restriction, on *line 8*, since it cannot create a non-continuous coarse grid solution from a continuous fine grid solution, i.e. if $\underline{v}^\ell \in Image(P_\ell)$ then $\underline{v}^L \in Image(P_L)$. Likewise the full weighting prolongation operator, used on *line 13*, will create a fine grid error whose fine level node values are linear interpolants of their parent node values. Hence, providing that $\underline{e}^L$ is contained in $Image(P_L)$ then the result of the prolongation, $\underline{e}^\ell$, will be contained in $Image(P_\ell)$.

The coarse grid right-hand side vector must be in the $Image(P_L^T)$, similar to the fine grid right-hand side vector. This is assured as components, $\underline{r}^L$ and $P_L^T \hat{K}^L(\underline{v}^L)$, can both be shown to be in $Image(P_L^T)$. The solution of the system defined on the coarse grid, $P_L^T \hat{K}^L(\underline{v}^L) = \underline{b}^L$, is performed using iterations of the Projected nonlinear Jacobi scheme and is therefore in $Image(P_L)$.

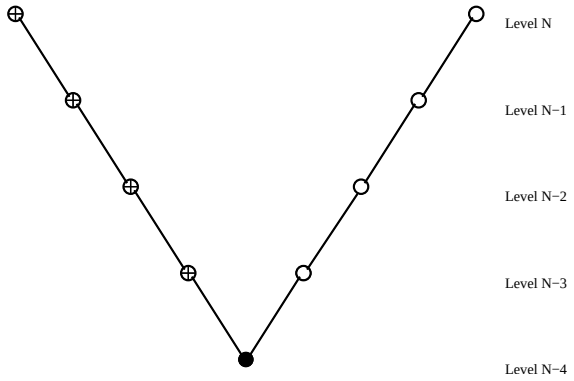


Figure 5.3: Schematic of a multigrid V-cycle: The V-cycle starts at the top left with a circle representing the initial operations on the finest grid. The information is passed down recursively to the coarsest grid (represented by the filled circle) and then passed back up to the finest grid (through each empty circle).

Finally we note that the measure used as a stopping criterion must be an accurate representation of the system being solved. For example, a residual norm calculated using a residual vector within the $Image(P_\ell)$:

$$\underline{r} = P_\ell[P_\ell^T \underline{\hat{b}} - \hat{K}\underline{u}].$$

5.5.2 The Full PFAS Scheme

The two-grid PFAS algorithm can be simply extended to give a recursive algorithm capable of using an arbitrary number of composite grid levels. The multigrid algorithm is given in Algorithm 12. In this case the grid levels of the projection operators are not given explicitly in the notation. The recursive multigrid version of the PFAS scheme has the same components as its two-grid counterpart. There are now two levels of iteration: the Projected Jacobi iterations employed for smoothing the solution error and the PFAS multigrid V-cycle iteration. Algorithm 12 (and indeed Algorithm 11) gives one step of a multigrid iterative solution process. In this case, one step is defined to be one cycle through all of the grid levels employed, starting at the most refined level. The structure

Algorithm 12 Function: Projected FAS Multigrid - V-Cycle

1: Input Parameter: l - level number of composite grid
2: Input Parameter: *base* - the coarsest level where smoothing is attempted
3: Input Parameter: $\underline{v}^\ell$ - initial solution guess such that $\underline{v}^\ell \in Image(P_\ell)$
4: Input Parameter: $\underline{b}^\ell$ - right-hand side vector such that $\underline{b}^\ell \in Image(P_\ell^T)$
5: **if** $\ell > base$ **then**
6: $L = \ell - 1$
7: Update $\underline{v}^\ell$ by smoothing ρ_1 times with the system $P_\ell^T \underline{\hat{K}}^\ell(\underline{u}^\ell) = \underline{b}^\ell$ using Projected Jacobi
8: Find residual $\underline{r}^\ell := \underline{b}^\ell - P_\ell^T \hat{K}^\ell(\underline{v}^\ell)$
9: Restrict residual $\underline{r}^L := \tilde{I}_\ell^L(\underline{r}^\ell)$
10: Restrict solution $\underline{v}^L := I_\ell^L(\underline{v}^\ell)$
11: Calculate coarse grid right-hand side $\underline{b}^L := \underline{r}^L + P_L^T \underline{\hat{K}}^L(\underline{v}^L)$
12: Save the initial coarse grid solution $\underline{v}^L_{init} = \underline{v}^L$
13: Re-call this function for level L, the system $P_L^T \underline{\hat{K}}^L(\underline{v}^L) = \underline{b}^L$ and the initial guess $\underline{v}^L_{init}$
14: Calculate the coarse grid correction $\underline{e}^L = \underline{v}^L - \underline{v}^L_{init}$
15: Prolongate the correction $\underline{e}^\ell := I_L^\ell(\underline{e}^L)$
16: Correct the fine grid solution $\underline{v}^\ell = \underline{v}^\ell + \underline{e}^\ell$
17: Update $\underline{v}^\ell$ by smoothing ρ_2 times with the system $P_\ell^T \underline{\hat{K}}^\ell(\underline{u}^\ell) = \underline{b}^\ell$ using Projected Jacobi
18: **else**
19: Smooth the system $P_\ell^T \underline{\hat{K}}^\ell(\underline{v}^\ell) = \underline{b}^\ell$ to a suitable tolerance
20: **end if**

of a V-cycle can be illustrated using Figure 5.3. Here, there are five composite grid levels used, ranging from the most refined level (N) to the coarsest level (N-4). The circle on the far left represents the operations on *lines 1-12* of Algorithm 12 applied to composite grid N. The line leaving this first circle represent *line 13* where the solution process is passed down to the next coarser grid (N-1). The illustration shows that the first half of the function is executed on all of the grids until the coarsest grid is reached (represented by the filled circle at the bottom of the V in Figure 5.3) where an exact solution is found. As the illustration implies, the correction is then passed up through all of the composite grid levels. For each grid, the operations on *lines 14-17* are executed.

We have shown that none of the operations in the two-grid PFAS remove the intermediate vectors from their respective vector spaces. This also applies to the multigrid version as the component operations are the same. Clearly, each composite grid level has a different number, and distribution, of points. Therefore System (5.5) needs to be defined for each composite grid level used.

5.6 Nonlinear Elliptic Model Problem

In order to test the performance of our new PFAS algorithm we first consider a scalar elliptic model problem of the form

$$-\overrightarrow{\nabla}^2 u + \gamma u e^u = f \tag{5.9}$$

on the domain $\Omega = (-1,1) \times (-1,1)$ and subject to $u = 0$ on the entire boundary. For the purposes of testing the source term, f, is chosen to permit the exact solution: $u = 1 - tanh(k(x^2 + y^2 - \alpha^2))$. This solution has a relatively small area of high curvature in the region of $x^2 + y^2 = \alpha^2$, making it an appropriate problem to evaluate the multigrid solver on highly refined grids. For the results shown here $\gamma = 1$, $k = 25$ and $\alpha = \frac{1}{2}$. Using

the continuous Galerkin FE method (introduced in Section 2.2.1) we produce a system of nonlinear equations equivalent to Equation (5.4). The component-wise definition of the operator vector is

$$[\underline{\hat{K}}(\underline{u})]_j = \sum_{e=1}^{ne} \left\{ \int_{\Omega_e} \overrightarrow{\nabla}\overline{u} \cdot \overrightarrow{\nabla} N_j d\Omega_e + \gamma \int_{\Omega_e} \overline{u} e^{\overline{u}} N_j d\Omega_e \right\}, \tag{5.10}$$

(where *ne* is the number of elements) and the same of the right-hand side vector is

$$\hat{b}_j = \sum_{e=1}^{ne} \left\{ \int_{\Omega_e} f N_j d\Omega_e \right\}. \tag{5.11}$$

In order to calculate the corrections defined above we must also calculate the Jacobi vector. This is defined (component-wise) as:

$$\frac{\partial \underline{\hat{K}}_j(\underline{u})}{\partial u_j} = \sum_{e=1}^{ne} \left\{ \int_{\Omega_e} \frac{\partial}{\partial u_j} (\overrightarrow{\nabla}\overline{u} \cdot \overrightarrow{\nabla} N_j) d\Omega_e + \gamma \int_{\Omega_e} \frac{\partial}{\partial u_j} (\overline{u} e^{\overline{u}}) N_j d\Omega_e \right\}.$$

The first differential term on the right-hand side can be written as:

$$\frac{\partial}{\partial u_j} (\overrightarrow{\nabla}\overline{u} \cdot \overrightarrow{\nabla} N_j) = \frac{\partial N_j}{\partial x} \frac{\partial}{\partial x} \left(\frac{\partial \overline{u}}{\partial u_j} \right) + \frac{\partial N_j}{\partial y} \frac{\partial}{\partial y} \left(\frac{\partial \overline{u}}{\partial u_j} \right).$$

Since $\overline{u} = \sum_{i=1}^{n} u_i N_i$, the differential, $\frac{\partial \overline{u}}{\partial u_j}$, is only non-zero when $i = j$. In this case $\frac{\partial u_j N_j}{\partial u_j} = N_j$. Therefore:

$$\frac{\partial}{\partial u_j} (\overrightarrow{\nabla}\overline{u} \cdot \overrightarrow{\nabla} N_j) = \left(\frac{\partial N_j}{\partial x} \right)^2 + \left(\frac{\partial N_j}{\partial y} \right)^2.$$

In order to differentiate the second differential term we employ the product rule giving

$$\frac{\partial}{\partial u_j} (\overline{u} e^{\overline{u}}) = \frac{\partial}{\partial u_j} (\overline{u}) e^{\overline{u}} + \overline{u} \frac{\partial}{\partial u_j} (e^{\overline{u}}).$$

Using the definition of $\overline{u}$ as above we get:

$$\frac{\partial}{\partial u_j}(\overline{u}e^{\overline{u}}) = N_j e^{\overline{u}} + \overline{u}N_j e^{u_j N_j}.$$

The resulting Jacobian vector entry is:

$$\frac{\partial K_j(\underline{u})}{\partial u_j} = \sum_{e=1}^{ne} \left\{ \int_{\Omega_e} \left(\frac{\partial N_j}{\partial x}\right)^2 + \left(\frac{\partial N_j}{\partial y}\right)^2 d\Omega_e + \gamma \int_{\Omega_e} (N_j e^{\overline{u}} + \overline{u}N_j e^{u_j N_j}) N_j d\Omega_e \right\}. \quad (5.12)$$

The three vectors (5.10), (5.11) and (5.12) are assembled by a loop over all of the elements. A contribution is calculated for each vertex of each element. Each of these vectors is then contained in a non-conforming space as the hanging nodes will receive contributions from only two elements rather than four.

5.6.1 Data Structures for the Solution Process

The data structures for the PFAS solution process are complicated by the need for many systems of various sizes. For both the solution and right-hand side we store a set of arrays. Each array in a set corresponds to one of the composite grids and is of the appropriate length for the number of points in that grid. No matrices are assembled during the PFAS solution process. If we were to use an assembled matrix $\hat{K}$ it would have to be re-formed every time a correction was made to the solution which would be a significant overhead.

The locally adaptive meshing data structures and algorithms introduced in Chapter 4 are used here with one modification. The vectors that are constructed for the Jacobi smoother have an entry for each internal point in the grid at each level on which that point exists. The assembly of these vectors for each of these points involves contributions from each of the four elements surrounding that point. If a point exists on more than one level of the grid then this set of surrounding elements may be different at each level. In order to speed

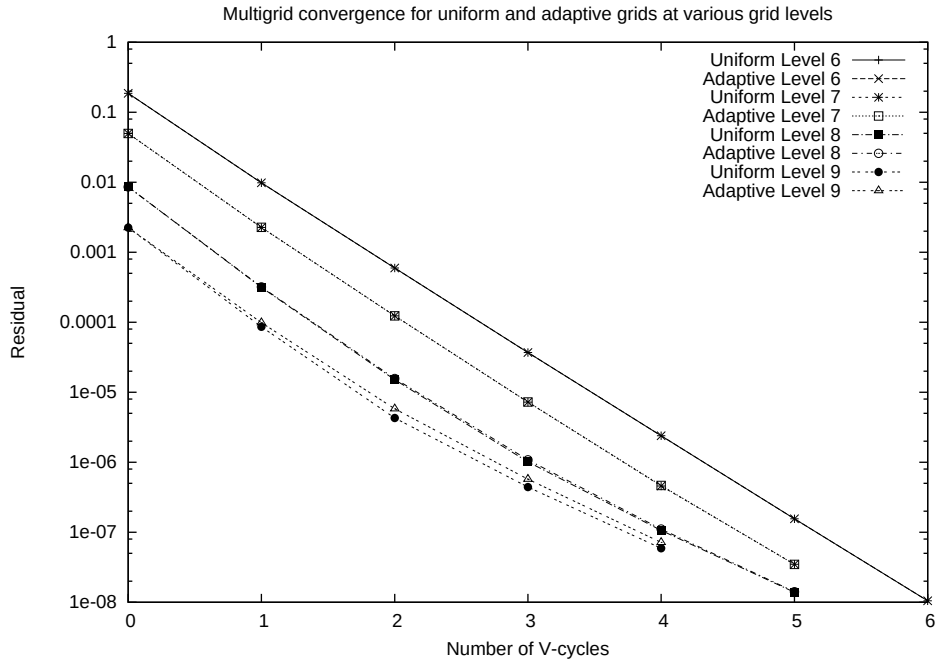


Figure 5.4: Elliptic model problem: Convergence of the PFAS: Multigrid convergence for uniform and adaptive grids at various levels.

up this assembly process an extra data structure was added which stores pointers to each of the four surrounding elements for each point on each level on which that point exists.

5.6.2 Results

In this section we show that the PFAS method performs as a standard FAS multigrid method would in terms of convergence, efficiency and accuracy, whilst also being able to solve problems defined on locally refined grids with hanging nodes.

Solver Convergence

The convergence of the multigrid solver for the nonlinear test problem can be seen in Figure 5.4. The norm of the residual is reduced by almost a constant factor at each v-cycle, for all choices of maximum grid refinement. There is also little difference in performance between globally refined (‘uniform’) and locally refined (‘adaptive’) grids.

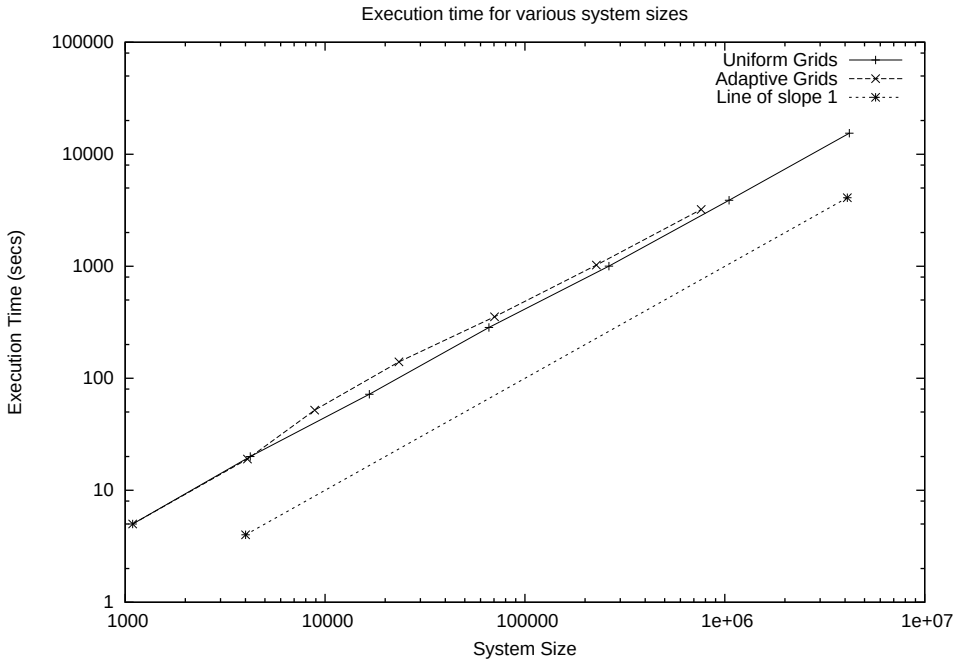


Figure 5.5: Elliptic model problem: Execution time against system size: This relationship is linear for both the uniform and locally refined cases

Work Required

Figure 5.5 shows that the execution time is linearly dependent on the system size. There is a slight overhead in the h-adaptive case for refinement and hanging node operations. However, this is far outweighed by the advantage of using less grid points.

Accuracy

Table 5.1 illustrates that for both uniformly and locally refined grids the error achieves the expected $O(h^2)$ behaviour. This table also shows the difference in system size between the uniform and adaptive cases. It seems that, using the new projected nonlinear Jacobi smoother, standard FAS multigrid performance can be replicated over non-uniform grids with hanging nodes.

Level	Uniform		Adaptive	
	No. Nodes	L_2-norm of Error	No. Nodes	L_2-norm of Error
4	289	2.0456×10^{-1}	289	2.0456×10^{-1}
5	1089	6.7681×10^{-2}	1089	6.7681×10^{-2}
6	4225	1.6865×10^{-2}	4093	1.6865×10^{-2}
7	16641	4.2772×10^{-3}	8873	4.2772×10^{-3}
8	66049	1.0739×10^{-3}	23425	1.0739×10^{-3}
9	263169	2.6878×10^{-4}	70485	2.6891×10^{-4}
10	1050625	6.7230×10^{-5}	227681	6.7730×10^{-5}

Table 5.1: Elliptic model problem: Solution accuracy: The error in the solution and the system size are given for various grid levels, in the uniform and adaptive cases.

5.7 Discussion

In this chapter we have integrated the adaptive meshing method introduced in Chapter 4 into an adaptive, finite element, multigrid solution method capable of solving nonlinear problems over grids containing hanging nodes. The basic mechanism of the standard Galerkin finite element assembly remains intact as a non-conforming system is built from individual element integrals. Projection methods are used within the Jacobi smoother to ensure that the resulting solution is continuous. The methods are straightforward to implement and effective. It has been shown that the PFAS scheme provides good convergence for a nonlinear elliptic model problem on a globally refined grid as well as showing that the use of locally refined grids does not degrade this convergence significantly. The method has also shown an error convergence (as Δx is reduced) on a locally refined grid that is equivalent to that on a globally refined grid.

The existing adaptive multigrid methods, some of which are described in Section 2.6, provide various alternative methods for dealing with the interfaces between sections of elements at different refinement levels. One of the most closely related methods to that presented here is the method of restricting the value of hanging nodes used in the FEARS package (described in Section 2.6). However, in that implementation the interpolation is integrated into the assembly of the element stiffness matrices, making this process more

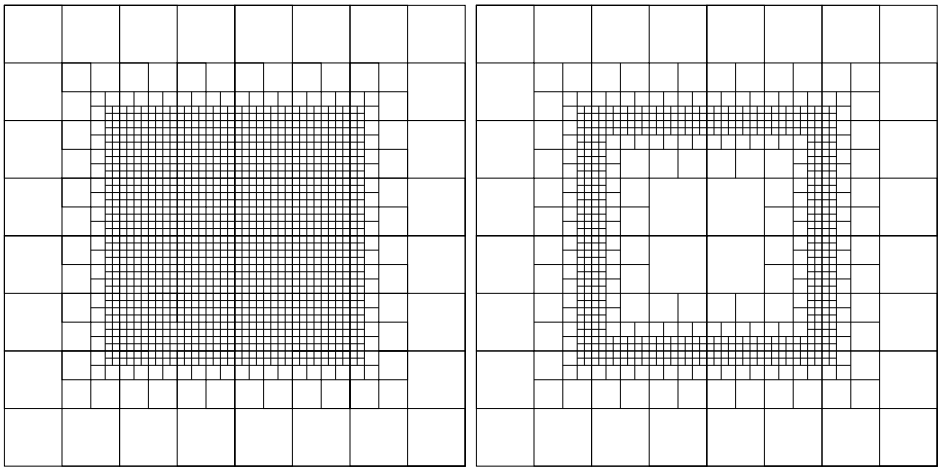


Figure 5.6: Typical patch and line refinement cases.

complex. The PFAS scheme avoids this complication by assembling the element matrices in the non-conforming space and keeping the restriction operations separate.

It is possible to criticise the PFAS scheme, in comparison to other adaptive multigrid algorithms (such as the MLAT and the FAC) as smooths are conducted over the whole domain at each level, meaning that our scheme may be less efficient. However, in any case of two-dimensional local refinement onto a line or block within the domain (e.g. Figure 5.6), the additional number of points added as an extra level of refinement is created is $\mathscr{O}(2n)$, (where n is the number of points on the previous level). In these common cases, therefore, our scheme is still optimal, and the overhead of smoothing over the entire domain becomes smaller (relatively) as the mesh is refined further.

Chapter 6

Projected FAS Multigrid for Parabolic Problems

6.1 Introduction

The purpose of this chapter is to demonstrate the successful application of the adaptive multigrid method, developed in Chapter 5, to the implicit solution of two representative nonlinear parabolic problems. Results are given for the Porous Medium Equation (PME) in Section 6.2. This equation, introduced in Section 3.3.1, has the form of a heat equation with a nonlinearity introduced into the diffusion term. The solutions considered here form a smooth dome shape which becomes flatter and wider with time. The locally refined mesh is therefore further refined around the edge of the dome as it advances into the domain. In Section 6.3 results are presented for the system of nonlinear parabolic equations adapted from the Burgers' equation, and introduced in Section 3.4. In this case the nonlinearity lies in the convection term. As with the PME the main solution features

propagate outwards from a central seed. In this case the advancing front has a small area of steep gradient which requires a fine locally refined grid to ensure that it is accurately resolved. Dynamic refinement is employed ahead of the front and coarsening behind it.

6.2 Porous Medium Equation

The porous medium equation (PME) is a parabolic equation with nonlinearity in the diffusion term. The governing equation is

$$\frac{\partial u}{\partial t} = \overrightarrow{\nabla} \cdot (u^m \overrightarrow{\nabla} u). \tag{6.1}$$

Using this model we will test the ability of the PFAS method to solve nonlinear parabolic problems, using implicit time-stepping, in an efficient and accurate manner. The method is applied to two variations of this problem. As mentioned in Chapter 3 the behaviour of the solution is affected by the choice of the parameter m. Solutions are found for the two cases, $m = 1$ and $m = 3$. The latter produces a steeper gradient in a particular, moving, region of the domain, thus making it a more troublesome problem to solve.

6.2.1 Similarity Solution

The results produced using our numerical scheme will be compared to a particular self-similar solution:

$$s(r,t) = \begin{cases} \frac{1}{(\lambda(t))^d}\left(1 - \left(\frac{r}{r_0\lambda(t)}\right)^2\right)^{\frac{1}{m}} & |r| \le r_0\lambda(t) \\ 0 & |r| > r_0\lambda(t) \end{cases}, \tag{6.2}$$

where

$$\lambda(t) = \left(\frac{t}{t_0}\right)^{\frac{1}{2+dm}} \tag{6.3}$$

and

$$t_0 = \frac{r_0^2 m}{2(2+dm)}. \tag{6.4}$$

The constants t_0 and r_0 represent the initial time and initial radius respectively. We choose $r_0 = 1$ from which, along with the dimension of our simulation (d=2), we can calculate, from (6.3), the initial time in terms of m:

$$t_0 = \frac{m}{4(1+m)}.$$

This also produces an alternative expression for $\lambda(t)$ in terms of m:

$$\lambda(t) = \left(\frac{4t(1+m)}{m}\right)^{\frac{1}{2(1+m)}}.$$

6.2.2 Initial Conditions and Boundary Conditions

The initial conditions are chosen to be the similarity solution at t_0. From Equation (6.3) $\lambda(t_0) = 1.0$ and so it follows that the initial conditions are:

$$u(r,t_0) = \begin{cases} (1-r^2)^{\frac{1}{m}} & |r| \leq 1.0 \\ 0 & |r| > 1.0 \end{cases}.$$

The two cases considered here are $m = 1$, giving $t_0 = \frac{1}{8}$, and $m = 3$, giving $t_0 = \frac{3}{16}$. Zero Dirichlet boundary conditions were employed around the entire boundary of the domain, which is taken to be $(-2,2) \times (-2,2)$. Note that this finite domain size means that the solution (6.2) is only valid until $\lambda(t) = 2$.

6.2.3 Discretisation

Using the FE discretisation techniques, introduced in Section 2.2.1, we discretise the spatial variables of equation (6.1) giving the weak form:

$$\int_\Omega \dot{\overline{u}} N_j d\Omega = -\int_\Omega \overline{u}^m \overrightarrow{\nabla}\overline{u} \cdot \overrightarrow{\nabla} N_j d\Omega$$

where $\overline{u} = \sum_{i=1}^{n} u_i N_i$ (where n is the number of interior nodes) and N_i and N_j are bilinear FE basis functions corresponding to nodes i and j respectively. This system of ordinary differential equations can then be presented in the $n \times n$ matrix form:

$$M\dot{\underline{u}} = -\underline{K}(\underline{u}).$$

Here M is the standard $n \times n$ mass matrix, $M_{ij} = \int_\Omega N_i N_j d\Omega$ and $\underline{K}$ the weighted stiffness vector, $K_j = \int_\Omega \overline{u}^m \overrightarrow{\nabla}\overline{u} \cdot \overrightarrow{\nabla} N_j d\Omega$. To complete the discretisation we now apply the second order trapezoidal rule (from Section 2.2.2). This is an implicit scheme which produces the following algebraic system at each time step

$$M\underline{u}^{k+1} + \frac{\Delta t}{2}\underline{K}(\underline{u}^{k+1}) = M\underline{u}^k - \frac{\Delta t}{2}\underline{K}(\underline{u}^k). \tag{6.5}$$

Equation (6.5) is a system of n nonlinear equations in the n unknowns $\underline{u}^{k+1}$.

6.2.4 Solution of Algebraic System

As with the implementations described for the elliptic problems in the previous chapter we need to clarify the form of the three vectors required for the Projected Jacobi iteration. Since the system of nonlinear equations is discretised over a locally refined grid, including hanging nodes, we can match this system to that in Equation (5.1). The three vectors that need to be calculated are therefore $\hat{\underline{b}}$, $\hat{K}(\underline{u}^{k+1})$ and $\frac{\partial \hat{K}}{\partial u_j}(\underline{u}^{k+1})$. For this problem these vectors are defined as follows:

Level	5	6	7	8	9	10
Time Step	2.0×10^{-2}	1.0×10^{-2}	5.0×10^{-3}	2.5×10^{-3}	1.25×10^{-3}	6.25×10^{-4}

Table 6.1: PME: Time step sizes: The choice of time step size is given for various choices of finest grid level, using the second order, trapezoidal, time stepping scheme.

$$\hat{b}_j = \sum_{e=1}^{ne}\left[\int_{\Omega_e}\left\{\overline{u}^k N_j - \frac{\Delta t}{2}(\overline{u}^k)^m\left[\frac{\partial N_j}{\partial x}\frac{\partial \overline{u}^k}{\partial x}+\frac{\partial N_j}{\partial y}\frac{\partial \overline{u}^k}{\partial y}\right]\right\}d\Omega_e\right],$$

$$[\hat{K}(\underline{u}^{k+1})]_j = \sum_{e=1}^{ne}\left[\int_{\Omega_e}\left\{\overline{u}^{k+1} N_j + \frac{\Delta t}{2}(\overline{u}^{k+1})^m\left[\frac{\partial N_j}{\partial x}\frac{\partial \overline{u}^{k+1}}{\partial x}+\frac{\partial N_j}{\partial y}\frac{\partial \overline{u}^{k+1}}{\partial y}\right]\right\}d\Omega_e\right]$$

and

$$\frac{\partial \hat{K}_j}{\partial u_j}(\underline{u}^{k+1}) = \sum_{e=1}^{ne}\left[\int_{\Omega_e}\left\{N_j^2 + m\frac{\Delta t}{2}N_j(\overline{u}^{k+1})^{m-1}\left[\frac{\partial N_j}{\partial x}\frac{\partial \overline{u}^{k+1}}{\partial x}+\frac{\partial N_j}{\partial y}\frac{\partial \overline{u}^{k+1}}{\partial y}\right]\right.\right.$$

$$\left.\left.+\frac{\Delta t}{2}(\overline{u}^{k+1})^m\left[\left(\frac{\partial N_j}{\partial x}\right)^2+\left(\frac{\partial N_j}{\partial x}\right)^2\right]\right\}d\Omega_e\right],$$

where *ne* is the number of elements in the grid.

6.2.5 Results

This section shows typical computational results for the PFAS method applied to the PME using bilinear finite elements and second order implicit time-stepping. Since the method is unconditionally stable there is no restriction on the step size from this perspective. For reasons of accuracy however we halve the step size each time the maximum refinement level is increased. The time step sizes used in the calculations that follow are given in Table 6.1.

A slice through of the solution in the $m = 1$ case can be seen in Figure 6.1 and with $m = 3$ in Figure 6.2. It is clearly evident that the latter case leads to a solution of higher

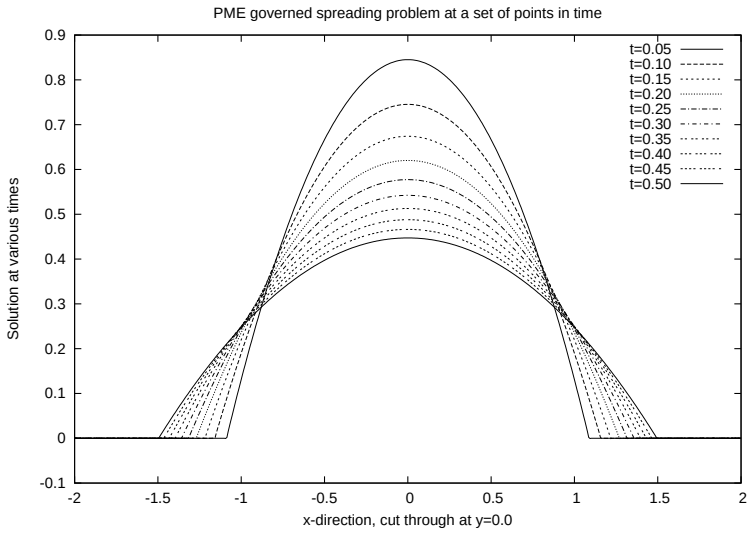


Figure 6.1: PME: Solution when m=1. This is a slice through the domain along the $y = 0$ line.

gradient in the contact region (note that the similarity solution has an infinite slope at $r = r_0\lambda(t)$ when $m = 3$). This in turn leads to a much greater numerical overshoot in the graph of the $m = 3$ solution in Figure 6.2, despite the use of intense local refinement. Two-dimensional views of a typical solution and grids are provided (for $m = 1$) in Figure 6.3.

The convergence of the residual with each PFAS V-Cycle is shown in Figure 6.4 for both the $m = 1$ and the $m = 3$ case. The convergence of the residual remains consistently good as the size of the smallest grid spacing is reduced. Although the $m = 3$ case is slightly less consistent both cases display the same rate of convergence, independent of the mesh size.

The convergence of the spatial error with grid refinement for both values of m is given in Figure 6.5. The error is reduced by an almost constant factor with each increase in maximum grid level. The $m = 3$ case is less accurate than the $m = 1$ case due to the increased overshoot apparent in Figure 6.2. This information is re-iterated in the second column of

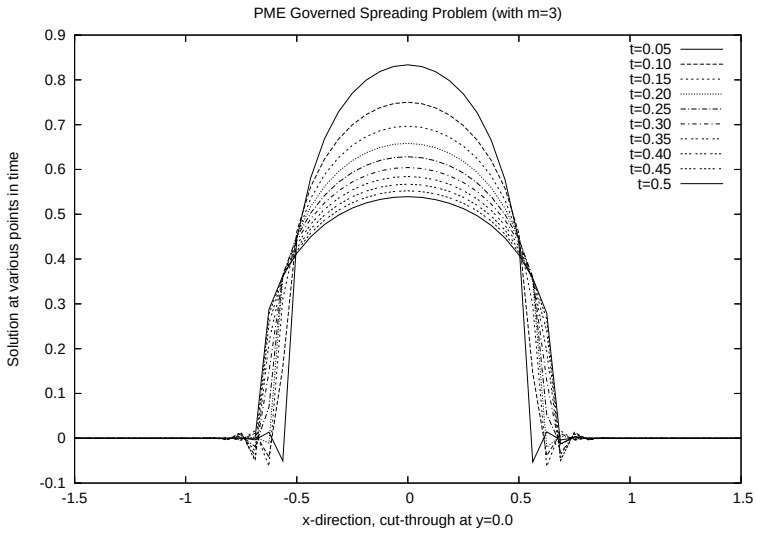


Figure 6.2: PME: Solution when m=3. This is a slice through the domain along the $y = 0$ line.

Tables 6.2 and 6.3. (The results given in these tables are for an absolute stopping tolerance which is less strict than the relative tolerance used for Figures 6.4 and 6.5.) Although the error is converging the behaviour is not $O(h^2)$ since second order convergence is only achieved for sufficiently smooth solutions. We know that for $m = 1$ the second derivative of $s(\overrightarrow{x})$ is unbounded when $r = r_0\lambda$. When $m = 3$ the first derivative of $s(\overrightarrow{x})$ is unbounded when $r = r_0\lambda$: which possibly explains the slower convergence. In the third column of Tables 6.2 and 6.3 the number of nodes are given for the first and the last time steps in the simulation demonstrating the increase in the size of the system caused by the increased refinement as the solution spreads. The total execution time is also given in each case. Figure 6.6 plots an average value of the number of nodes for each level against the execution time. This relationship can be seen to be roughly linearly dependent giving our algorithm a cost (in terms of CPU time) of $\mathcal{O}(n)$. Computational effort is saved, in comparison to a uniformly refined grid, by having a coarser grid outside the areas of high activity but the multigrid behaviour is retained. Of course the results in Tables 6.2 and 6.3 and Figures 6.1 and 6.2 are for discrete points in time.

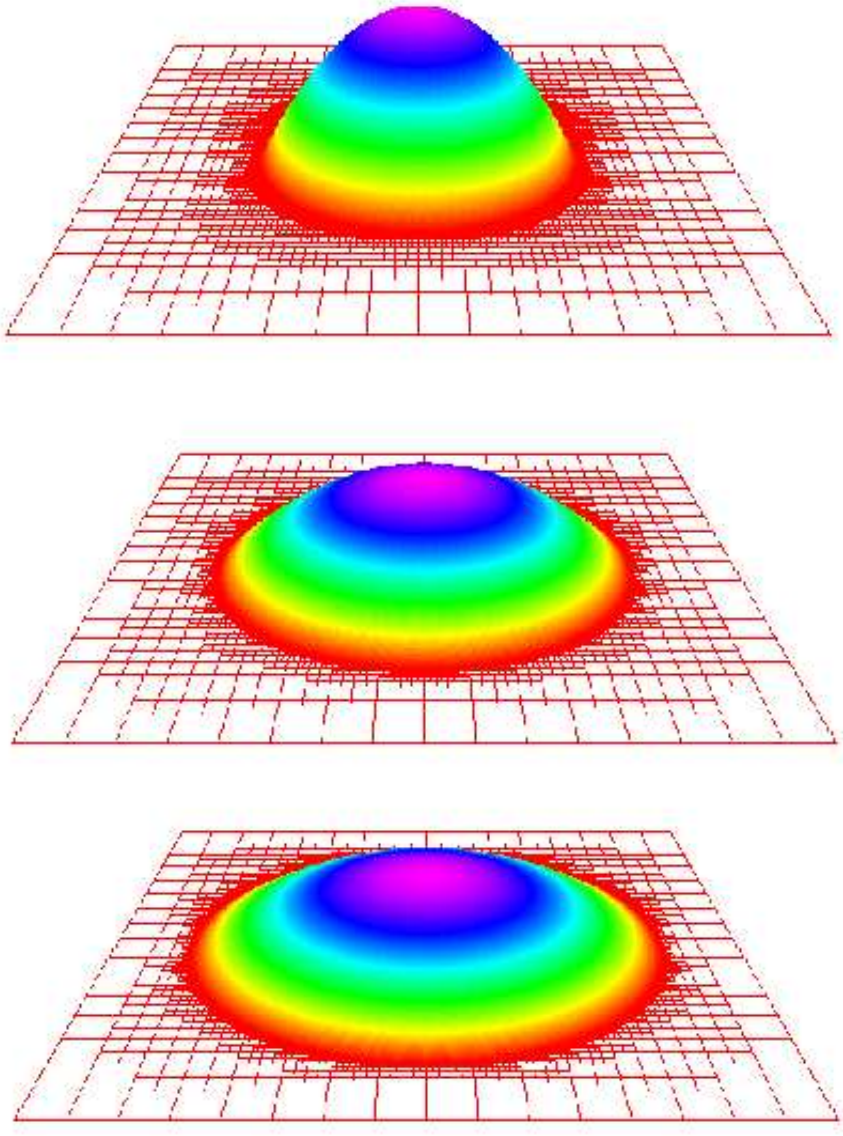


Figure 6.3: PME: Two-dimensional illustration of solution and grid (for $m = 1$) at steps, 10, 50 and 1000. i.e. t=0.005, t=0.250 and t=0.500. The maximum grid refinement level for these results is 8.

Level	$\lvert \underline{u} - \underline{s} \rvert_{L_2}$	No. Nodes	V-cycles	Time
5	3.2590×10^{-2}	609-981	7/8	6
6	1.6702×10^{-2}	1213-1685	6/7	23
7	5.8133×10^{-3}	2613-2933	4/5	66
8	2.0991×10^{-3}	6517-6837	3	211
9	8.1741×10^{-4}	19065-19865	2	783
10	3.2870×10^{-4}	63289-66947	1	2767

Table 6.2: PME: Results for the $m = 1$ case. The stopping tolerance is 10^{-8} and all values were taken at $t = 0.04$.

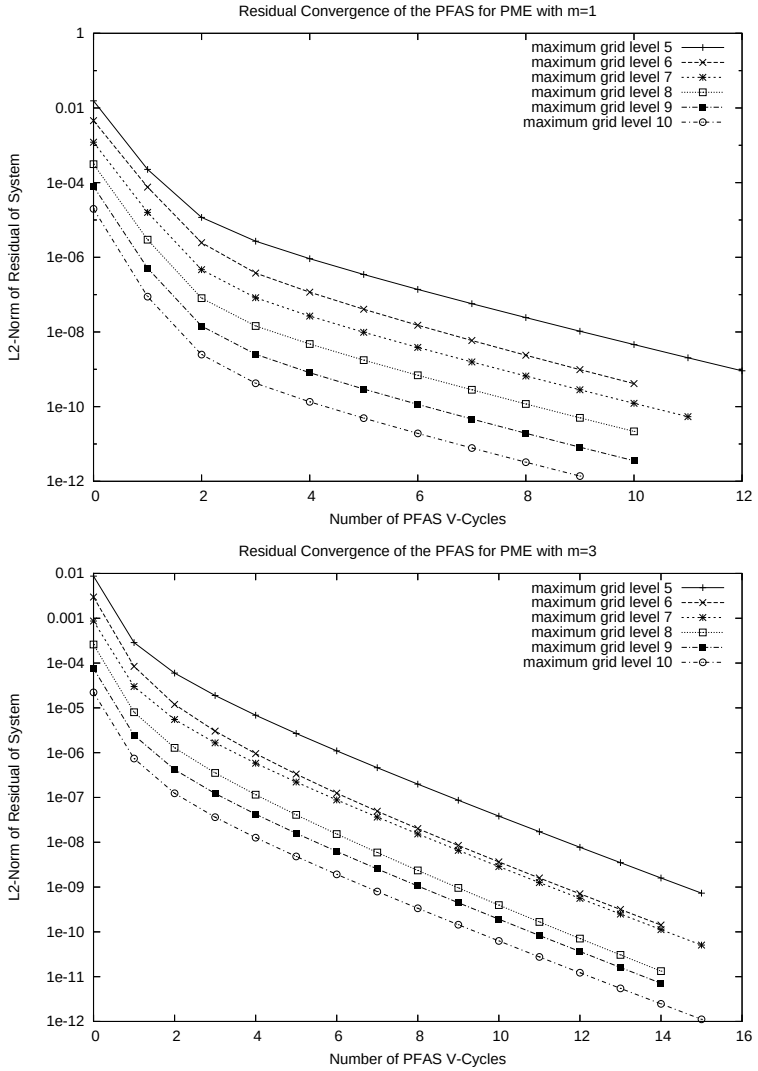


Figure 6.4: PME: Convergence of the PFAS method for the m=1 and m=3 case of the PME. PFAS v-cycles were performed until the residual was below a relative tolerance of 10^{-8}. (This tolerance is chosen to be strict for the purposes of testing.)

Level	$\lvert\underline{u}-\underline{s}\rvert_{L_2}$	No. Nodes	V-cycles	Time
5	1.5915×10^{-1}	609-1041	10	9
6	1.0016×10^{-1}	1213-1845	7	28
7	6.6500×10^{-2}	2613-2957	5	111
8	2.8364×10^{-2}	6517-6999	5	425
9	1.7624×10^{-2}	19065-19673	4	1908
10	1.0017×10^{-2}	63653-64737	2	7892

Table 6.3: PME: Results for the $m = 3$ case. The stopping tolerance is 10^{-8} and all values were taken at $t = 0.04$.

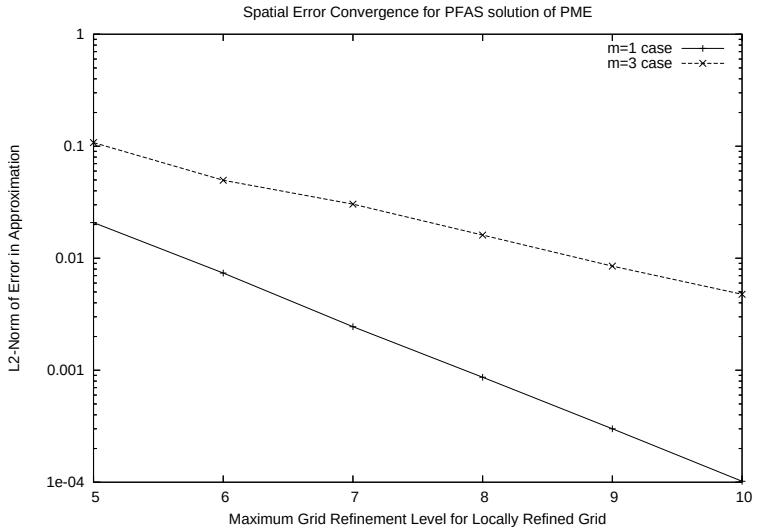


Figure 6.5: PME: Spatial error convergence for the m=1 and m=3 cases of the PME.

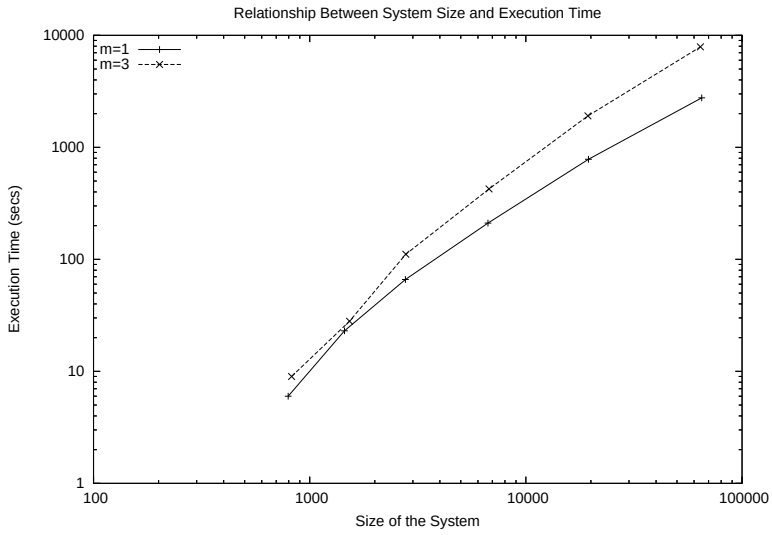


Figure 6.6: PME: Execution time against system size: The relationship between the number of internal nodes in the grid (the system size) and the execution time for the program (measured in seconds). The points plotted each correspond to a solution at a different maximum grid refinement level. The number of nodes is an average across the whole solution.

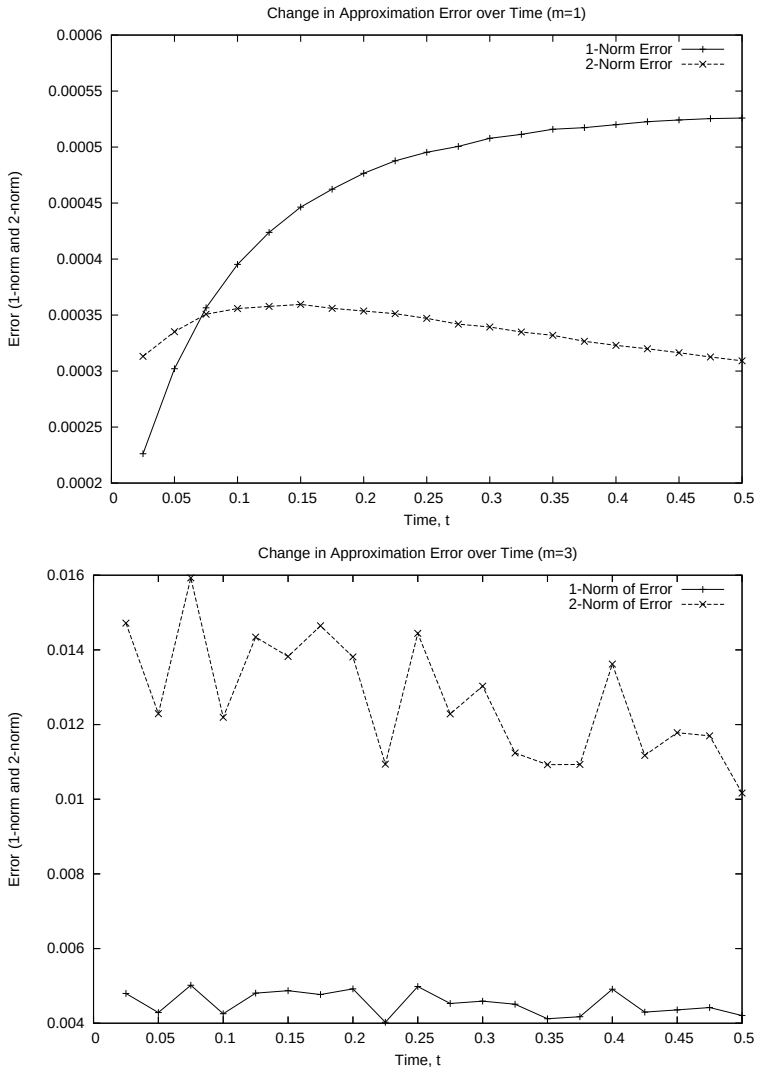


Figure 6.7: PME: Behaviour of error in the approximation of the PME solution (with $m = 1$ above and $m = 3$ below) over time.

The error in the approximation to the PME solution will also depend on time however. When testing against a similarity solution we might expect to see an increase in error over time. The L_1 and L_2 norms of the error, over time, are shown in Figure 6.7 (for $m = 1$ and $m = 3$, on a level 8 grid). The $m = 1$ graphs do indeed display an initial increase in error through time, however this lessens as the simulation continues and appears to eventually become constant as the similarity solution spreads more and more slowly. The $m = 3$ graphs are predictably more variable but equally do not show any consistent increase in error over time.

Finally we note that in the previous chapter it was demonstrated that, for elliptic problems, the computational work required for the PFAS method is proportional to the size of the system. This can be re-confirmed for this parabolic problem by comparing the number of nodes with the execution time in Table 6.2 (with $m = 1$). The equivalent results for the $m = 3$ case are given in Table 6.3 although here the execution times are longer as the convergence is slower.

6.3 A System of Burgers' Equations

In this section we extend the application of the PFAS method to *systems* of nonlinear parabolic equations. The example used here is the system of Burgers' equations introduced in Section 3.4.1. The nonlinearity of these two equations is in the convection term as opposed to the previous example where the nonlinearity lay in the diffusion term. The system of PDEs is given by:

$$\frac{\partial u}{\partial t} = \frac{-v}{r}(\overrightarrow{x} \cdot \overrightarrow{\nabla} u) + \delta \overrightarrow{\nabla}^2 u \tag{6.6}$$

$$\frac{\partial v}{\partial t} = \frac{u}{r}(\overrightarrow{x} \cdot \overrightarrow{\nabla} v) + \delta \overrightarrow{\nabla}^2 v, \tag{6.7}$$

where $\delta > 0$. The equations (6.6) and (6.7) are again discretised using bilinear finite elements combined with the backwards Euler method, and also with the trapezoidal rule, in time. The purpose of this implementation is to test the performance of PFAS on a coupled system of parabolic PDEs, as well as to evaluate whether a small highly refined region of the grid, which is redefined over time, is detrimental to the accuracy of the solution under this scheme. The problem is defined over a $(-1,1) \times (-1,1)$ domain with zero Dirichlet conditions on the entire boundary.

6.3.1 Discretisation

A FE discretisation in space gives the weak form:

$$\int_\Omega \dot{\overline{u}} N_j d\Omega + \int_\Omega \frac{-\overline{v}}{r}(\overrightarrow{x}\cdot\overrightarrow{\nabla}\overline{u})N_j d\Omega = -\delta \int_\Omega \underline{\nabla}\overline{u}\cdot\overrightarrow{\nabla}N_j d\Omega,$$

$$\int_\Omega \dot{\overline{v}} N_j d\Omega + \int_\Omega \frac{\overline{u}}{r}(\overrightarrow{x}\cdot\overrightarrow{\nabla}\overline{v})N_j d\Omega = -\delta \int_\Omega \underline{\nabla}\overline{v}\cdot\overrightarrow{\nabla}N_j d\Omega$$

for $j = 1,...,n$. Again $\overline{u} = \sum_{i=1}^{n} u_i N_i$ and $\overline{v} = \sum_{i=1}^{n} v_i N_i$ and n is the number of interior points. The discretised system can be represented in the matrix form:

$$M\dot{\underline{u}} + \underline{G}(\underline{u}, -\underline{v}) = -\delta K\underline{u},$$

$$M\dot{\underline{v}} + \underline{G}(\underline{v}, \underline{u}) = -\delta K\underline{v},$$

where M and K are the standard mass and stiffness matrices respectively and $\underline{\mathrm{G}}$ is a vector built using the nonlinear term which, in general form, is given by:

$$G_j(\underline{a}, \underline{b}) = \int_\Omega \frac{\overline{b}}{r}(\overrightarrow{x}\cdot\overrightarrow{\nabla}\overline{a})N_j d\Omega,$$

for $j = 1,...,n$. Application of the backward Euler time stepping scheme gives:

$$M\underline{u}^{k+1} + \Delta t\underline{G}(\underline{u}^{k+1}, -\underline{v}^{k+1}) + \Delta t\delta K\underline{u}^{k+1} = M\underline{u}^{k},$$

$$M\underline{v}^{k+1} + \Delta t\underline{G}(\underline{v}^{k+1}, \underline{u}^{k+1}) + \Delta t\delta K\underline{v}^{k+1} = M\underline{v}^{k}. \tag{6.8}$$

Application of the trapezoidal rule would give the slightly more complex pair of equations:

$$M\underline{u}^{k+1} + \frac{\Delta t}{2}\underline{G}(\underline{u}^{k+1}, -\underline{v}^{k+1}) + \frac{\Delta t}{2}\delta K\underline{u}^{k+1} = M\underline{u}^{k} - \frac{\Delta t}{2}\underline{G}(\underline{u}^{k}, -\underline{v}^{k}) - \frac{\Delta t}{2}\delta K\underline{u}^{k},$$

$$M\underline{v}^{n+1} + \frac{\Delta t}{2}\underline{G}(\underline{v}^{k+1}, \underline{u}^{k+1}) + \frac{\Delta t}{2}\delta K\underline{v}^{k+1} = M\underline{v}^{k} - \frac{\Delta t}{2}\underline{G}(\underline{v}^{k}, \underline{u}^{k}) - \frac{\Delta t}{2}\delta K\underline{v}^{k}, \tag{6.9}$$

but these would still be solved in essentially the same manner as described below.

6.3.2 Implementation

Coupling the System

System (6.8) is coupled by iteratively improving the approximation to each system alternately. In this way the equations can remain separate and may be smoothed independently. However, taking a sweep of the Projected Jacobi smoother over the approximation to Equation (6.6), then a sweep over the approximation to Equation (6.7) immediately afterwards, effectively couples the system. This approach is referred to in this thesis as a 'semi-coupled' iterative smoothing scheme.

Vectors for the Jacobi Smoother

The pair of coupled nonlinear systems of Equations (6.8) (similarly (6.9)) represent a discretisation over a locally refined grid containing hanging nodes. The pair of systems can be represented as

$$\hat{\underline{K}}(\underline{u}^{k+1}, \underline{v}^{k+1}) = \hat{\underline{b}}, \tag{6.10}$$

$$\hat{\underline{R}}(\underline{u}^{k+1}, \underline{v}^{k+1}) = \hat{\underline{d}}. \tag{6.11}$$

Since we will perform the Jacobi smooths over the two systems of equations separately, we must also define the three vectors, necessary for the iteration, separately. We recall from Section 5.6 that, in the case of a single PDE, these three vectors are $\hat{\underline{b}}$, $\hat{K}(\underline{u}^{k+1})$ and $\frac{\partial \hat{K}}{\partial u_j}(\underline{u}^{k+1})$. In this case, where each depends on two independent variables, these vectors take a slightly different form. The vectors are defined here in a component-wise manner. For simplicity, only the vectors corresponding to the backward Euler scheme are given.

The right-hand side vectors are:

$$\hat{b}_j = \sum_{e=1}^{ne} \left[\int_{\Omega_e} \overline{u}^k N_j d\Omega_e \right],$$

$$\hat{d}_j = \sum_{e=1}^{ne} \left[\int_{\Omega_e} \overline{v}^k N_j d\Omega_e \right].$$

The vectors resulting from the left-hand side integrals are:

$$[\underline{\hat{K}}(\underline{u}^{k+1}, \underline{v}^{k+1})]_j = \sum_{e=1}^{ne} \left[\int_{\Omega_e} \left\{ \overline{u}^{k+1} N_j - \frac{\Delta t \overline{v}^{k+1} N_j}{r} \left[x \frac{\partial \overline{u}^{k+1}}{\partial x} + y \frac{\partial \overline{u}^{k+1}}{\partial y} \right] \right.\right.$$

$$\left.\left. + \Delta t \delta \left[\frac{\partial \overline{u}^{k+1}}{\partial x} \frac{\partial N_j}{\partial x} + \frac{\partial \overline{u}^{k+1}}{\partial y} \frac{\partial N_j}{\partial y} \right] \right\} d\Omega_e \right],$$

$$[\underline{\hat{R}}(\underline{u}^{k+1}, \underline{v}^{k+1})]_j = \sum_{e=1}^{ne} \left[\int_{\Omega_e} \left\{ \overline{v}^{k+1} N_j + \frac{\Delta t \overline{u}^{k+1} N_j}{r} \left[x \frac{\partial \overline{v}^{k+1}}{\partial x} + y \frac{\partial \overline{v}^{k+1}}{\partial y} \right] \right.\right.$$

$$\left.\left. + \Delta t \delta \left[\frac{\partial \overline{v}^{k+1}}{\partial x} \frac{\partial N_j}{\partial x} + \frac{\partial \overline{v}^{k+1}}{\partial y} \frac{\partial N_j}{\partial y} \right] \right\} d\Omega_e \right].$$

The Jacobian vectors are therefore:

$$\frac{\partial \hat{K}_j}{\partial u_j}(\underline{v}^{k+1}) = \sum_{e=1}^{ne} \left[\int_{\Omega_e} \left\{ N_j^2 - \frac{\Delta t \overline{v}^{k+1} N_j}{r} \left[x \frac{\partial N_j}{\partial x} + y \frac{\partial N_j}{\partial y} \right] + \Delta t \delta \left[\left(\frac{\partial N_j}{\partial x} \right)^2 + \left(\frac{\partial N_j}{\partial y} \right)^2 \right] \right\} d\Omega_e \right],$$

$$\frac{\partial \hat{R}_j}{\partial v_j}(\underline{u}^{k+1}) = \sum_{e=1}^{ne} \left[\int_{\Omega_e} \left\{ N_j^2 + \frac{\Delta t \overline{u}^{k+1} N_j}{r} \left[x \frac{\partial N_j}{\partial x} + y \frac{\partial N_j}{\partial y} \right] + \Delta t \delta \left[\left(\frac{\partial N_j}{\partial x} \right)^2 + \left(\frac{\partial N_j}{\partial y} \right)^2 \right] \right\} d\Omega_e \right].$$

Multigrid V-Cycle for Two Coupled Equations

The adaptation of the PFAS method to allow for the coupling of two algebraic systems is given in Algorithm 13. All of the vectors (of length n) required for the original PFAS V-cycle algorithm given in Algorithm 12 (in Section 5.5) are duplicated for the second independent variable in this case. Lines *10-14*, *25-29* and *31-36* are examples of the semi-coupled iterative smoothing method described briefly above. For any one level the

systems to be smoothed are:

$$P^T \underline{\hat{K}}(\underline{u}^{(r+1)}, \underline{v}^{(r)}) = \underline{b}, \tag{6.12}$$

$$P^T \underline{\hat{R}}(\underline{u}^{(r+1)}, \underline{v}^{(r+1)}) = \underline{d}. \tag{6.13}$$

where $\underline{v}^{(r)}$ is the approximation of resulting from the r^{th} smooth and $\underline{v}^{(r+1)}$ and $\underline{u}^{(r+1)}$ are the approximations of resulting from the $(r+1)^{th}$ smooth. The vectors $\underline{b}$ and $\underline{v}^{(r)}$ are constant during the smooth over (6.12). Similarly, $\underline{d}$ and $\underline{u}^{(r+1)}$ are constant during the smooth over (6.13). The latest version of the approximate solution is used wherever possible. The smooth over system (6.12) uses $\underline{v}^{(r)}$ from the previous smooth where the smooth over system (6.12) uses $\underline{u}^{(r+1)}$ from the current smooth. (This approach is explained in a little more depth in Chapter 7.)

6.3.3 Results

This section demonstrates that the PFAS method can effectively solve coupled systems of nonlinear parabolic equations over locally highly refined grids. Since the solution contains small transient regions of high gradient, not only is the grid heavily refined over a small area but the grid also changes with time as the solution progresses. We have shown in Chapter 4 that this form of adaptivity is able to maintain accuracy to a reasonable level while significantly reducing the computational costs. A typical grid produced for this application is shown in Figure 6.8 where $rtol = 10^{-3}$, $ctol = 10^{-4}$. Figure 6.9 shows the a cut-through section of the solution at various points in time. We can see that the refinement and de-refinement has changed the mesh significantly between the time steps shown.

The size of time steps taken for various finest grid levels are given in Table 6.4. These are different for the two time discretisation schemes. Since the trapezoidal rule is second order we need only halve the time step size when the next grid level is used. For the first

Algorithm 13 Function: PFAS V-Cycle for Two Coupled Equations

Input Parameter: l - level number of composite grid
Input Parameter: *base* - the coarsest level on which smoothing is attempted
Input Parameter: $\underline{u}^\ell$ - initial solution guess such that $\underline{u}^\ell \in Image(P_\ell)$
Input Parameter: $\underline{v}^\ell$ - initial solution guess such that $\underline{v}^\ell \in Image(P_\ell)$
Input Parameter: $\underline{b}^\ell$ - right-hand side vector (in terms of $\underline{u}^\ell$) such that $\underline{b}^\ell \in Image(P_\ell^T)$
Input Parameter: $\underline{d}^\ell$ - right-hand side vector (in terms of $\underline{v}^\ell$) such that $\underline{d}^\ell \in Image(P_\ell^T)$
if $\ell > base$ **then**
  $L = \ell - 1$
  $count = 0$
  while $count < \rho_1$ **do**
    Update $\underline{u}^\ell$ by smoothing with the system: $P_\ell^T \hat{K}^\ell(\underline{u}^\ell, \underline{v}^\ell) = \underline{b}^\ell$ using PJ
    Update $\underline{v}^\ell$ by smoothing with the system: $P_\ell^T \hat{R}^\ell(\underline{u}^\ell, \underline{v}^\ell) = \underline{d}^\ell$ using PJ
    $count++$
  end while
  Find residuals $\underline{r}_u^\ell := \underline{b}^\ell - P_\ell^T \hat{K}^\ell(\underline{u}^\ell, \underline{v}^\ell)$ and $\underline{r}_v^\ell := \underline{d}^\ell - P_\ell^T \hat{R}^\ell(\underline{u}^\ell, \underline{v}^\ell)$
  Restrict residuals $\underline{r}_u^L := \tilde{I}_\ell^L(\underline{r}_u^\ell)$ and $\underline{r}_v^L := \tilde{I}_\ell^L(\underline{r}_v^\ell)$
  Restrict solutions $\underline{u}^L := I_\ell^L(\underline{u}^\ell)$ and $\underline{v}^L := I_\ell^L(\underline{v}^\ell)$
  Calculate coarse grid right-hand sides $\underline{b}^L := \underline{r}_u^L + P_L^T \hat{K}^L(\underline{u}^L, \underline{v}^L)$
  and $\underline{d}^L := \underline{r}_v^L + P_L^T \hat{R}^L(\underline{u}^L, \underline{v}^L)$
  Save the initial coarse grid solutions $\underline{u}_{init}^L = \underline{u}^L$ and $\underline{v}_{init}^L = \underline{v}^L$
  Re-call this function for level L with the systems $P_L^T \hat{K}^L(\underline{u}^L, \underline{v}^L) = \underline{b}^L$ and $P_L^T \hat{R}^L(\underline{u}^L, \underline{v}^L) = \underline{d}^L$ with the initial guesses $\underline{u}_{init}^L$ and $\underline{v}_{init}^L$
  Calculate the coarse grid corrections $\underline{e}_u^L = \underline{u}^L - \underline{u}_{init}^L$ and $\underline{e}_v^L = \underline{v}^L - \underline{v}_{init}^L$
  Prolongate the corrections $\underline{e}_u^\ell := I_L^\ell(\underline{e}_u^L)$ and $\underline{e}_v^\ell := I_L^\ell(\underline{e}_v^L)$
  Correct the fine grid solution $\underline{u}^\ell = \underline{u}^\ell + \underline{e}_u^\ell$ and $\underline{v}^\ell = \underline{v}^\ell + \underline{e}_v^\ell$
  $count = 0$
  while $count < \rho_2$ **do**
    Update $\underline{u}^\ell$ by smoothing with the system: $P_\ell^T \hat{K}^\ell(\underline{u}^\ell, \underline{v}^\ell) = \underline{b}^\ell$ using PJ
    Update $\underline{v}^\ell$ by smoothing with the system: $P_\ell^T \hat{R}^\ell(\underline{u}^\ell, \underline{v}^\ell) = \underline{d}^\ell$ using PJ
    $count++$
  end while
else
  Find residuals $\underline{r}_u^\ell := \underline{b}^\ell - P_\ell^T \hat{K}^\ell(\underline{u}^\ell, \underline{v}^\ell)$ and $\underline{r}_v^\ell := \underline{d}^\ell - P_\ell^T \hat{R}^\ell(\underline{u}^\ell, \underline{v}^\ell)$
  while $|\underline{r}_u^\ell|_{L_2} > ntol$ and $|\underline{r}_v^\ell|_{L_2} > ntol$ **do**
    Smooth the system: $P_\ell^T \hat{K}^\ell(\underline{u}^\ell, \underline{v}^\ell) = \underline{b}^\ell$
    Smooth the system: $P_\ell^T \hat{R}^\ell(\underline{u}^\ell, \underline{v}^\ell) = \underline{d}^\ell$
    Find residuals $\underline{r}_u^\ell := \underline{b}^\ell - P_\ell^T \hat{K}^\ell(\underline{u}^\ell, \underline{v}^\ell)$ and $\underline{r}_v^\ell := \underline{d}^\ell - P_\ell^T \hat{R}^\ell(\underline{u}^\ell, \underline{v}^\ell)$
  end while
end if

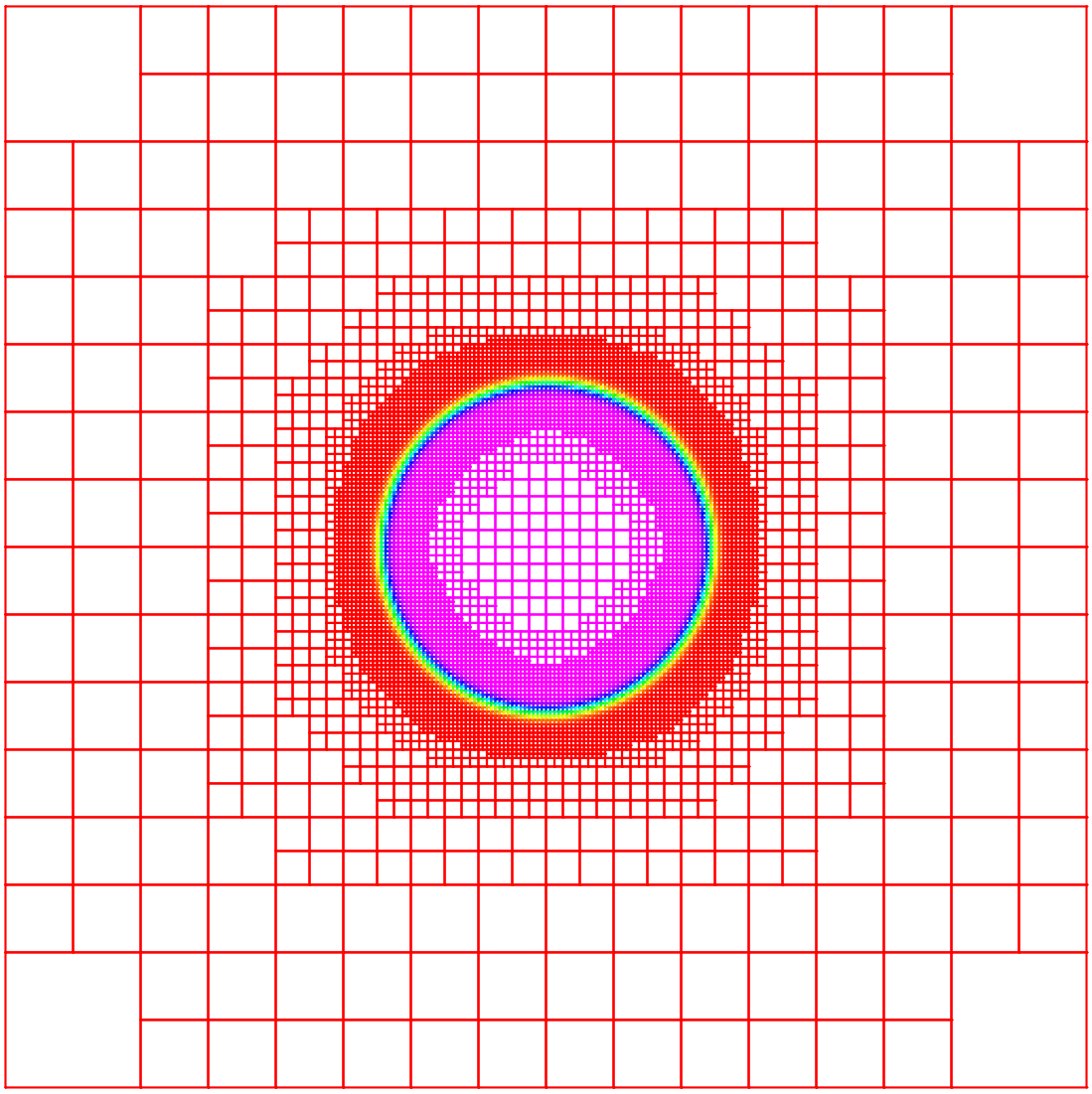


Figure 6.8: System of Burgers' equations: A typical grid for the solution of the system of coupled Burgers' equations. The change in colour denotes the change in the u value.

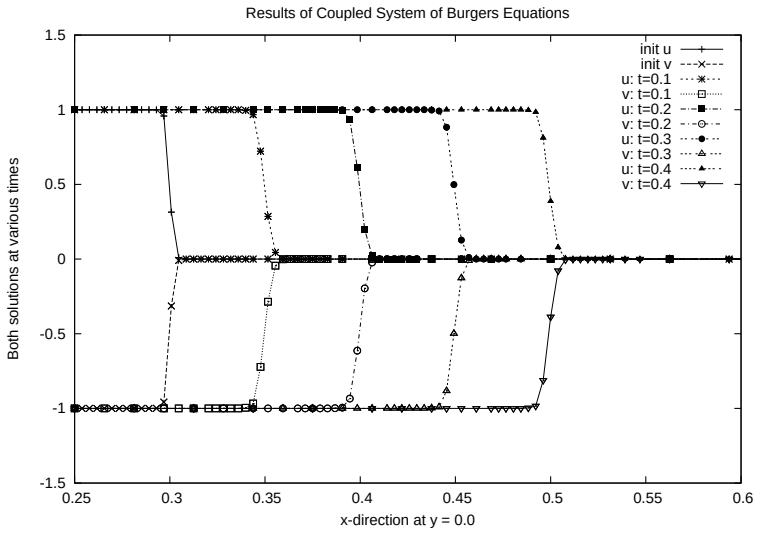


Figure 6.9: System of Burgers' equations: Solutions for u and v at various time steps. The grid points are shown for all solutions.

Level	7	8	9	10
Backward Euler	2.0×10^{-3}	5.0×10^{-4}	1.25×10^{-4}	3.125×10^{-5}
Trapezoidal	2.0×10^{-3}	1.0×10^{-3}	5.0×10^{-4}	2.5×10^{-4}

Table 6.4: System of Burgers' equations: The time step sizes which were used for four choices of finest grid level in the case of the backward Euler method and of the trapezoidal rule.

order time stepping Δt must be quartered in order to maintain accuracy. Stability is never an issue for any of the results presented due to the implicit nature of the two schemes.

Figure 6.10 contains convergence results for the PFAS multigrid method. The stopping criterion used for these results was a relative tolerance of 10^{-8} for the L_2-norm of the residual of the system. This is unnecessarily cautious for most applications but was used to fully demonstrate the convergence. There is no deterioration in the multigrid performance as the grid is further refined.

As described above, the solution of a pair of coupled PDEs requires the solution of a system of $2n$ nonlinear algebraic equations in $2n$ unknowns at each implicit time step. Furthermore, for this particular problem, as the solve progresses the area of high gradi-

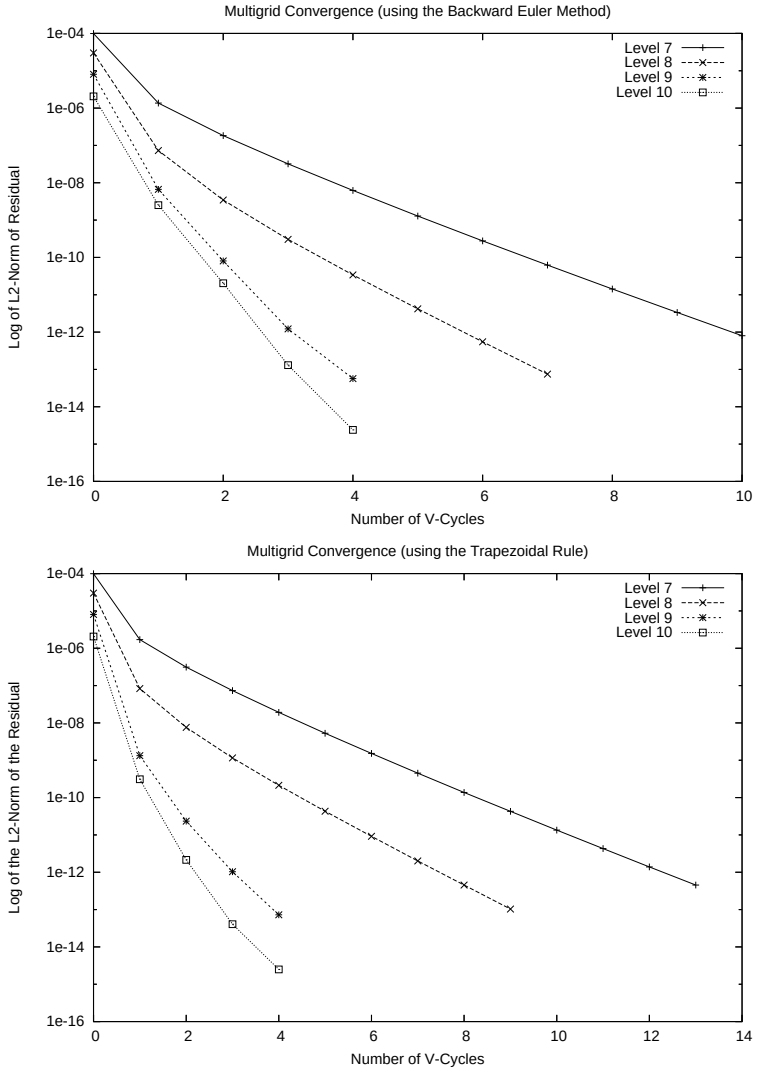


Figure 6.10: System of Burgers' equations: The PFAS multigrid convergence for four choices of finest grid level.

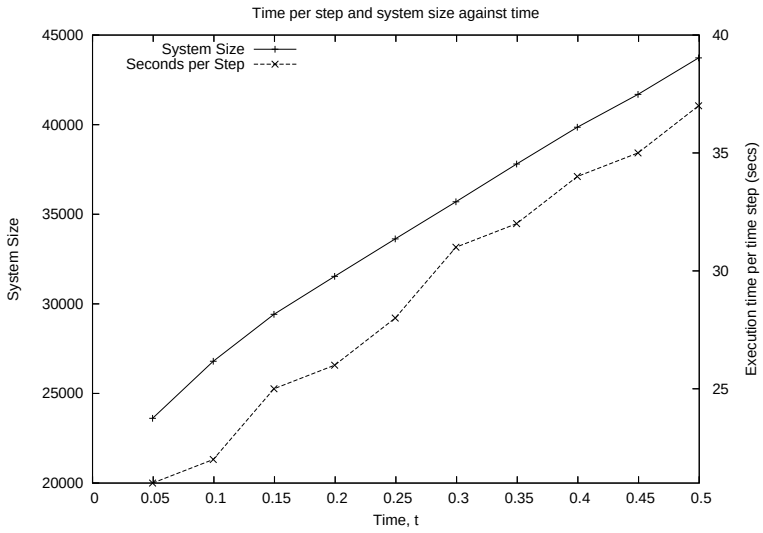


Figure 6.11: System of Burgers' equations: The size of the system and the execution time per step are plotted against the time, t, showing the correspondence between their growth patterns.

ent becomes larger. The number of finest level elements therefore increases (in a similar manner to the PF model simulation in Chapter 4) and consequently so does the system size. The increase in system size, along with the corresponding increase in execution time per step is illustrated in Figure 6.11. The correspondence between the two is still strong in this case: demonstrating that the PFAS algorithm is still optimal (i.e. the cost is proportional to n) for this problem.

Since we do not know an exact analytical solution for this problem we cannot compute exact errors, as with the PME in the previous section. For this problem we therefore compare the solutions produced with different levels of refinement. If $\underline{\omega}^l$ is the approximate solution produced on a grid with a maximum refinement level of l and $\underline{\omega}^{l-1}$ is the approximation produced on the grid with a maximum refinement level of $l-1$ then the difference is $\underline{e}^{l,l-1} = \underline{\omega}^{l-1} - \underline{\omega}^l$. These differences may be used as an approximation to the spatial discretisation error. We expect to see the difference between $\underline{\omega}^l$ and $\underline{\omega}^{l-1}$ reduce by a uniform factor as l increases. The difficulty of this approach however is that we must compare solutions that are defined on grids with differing sets of points. In order

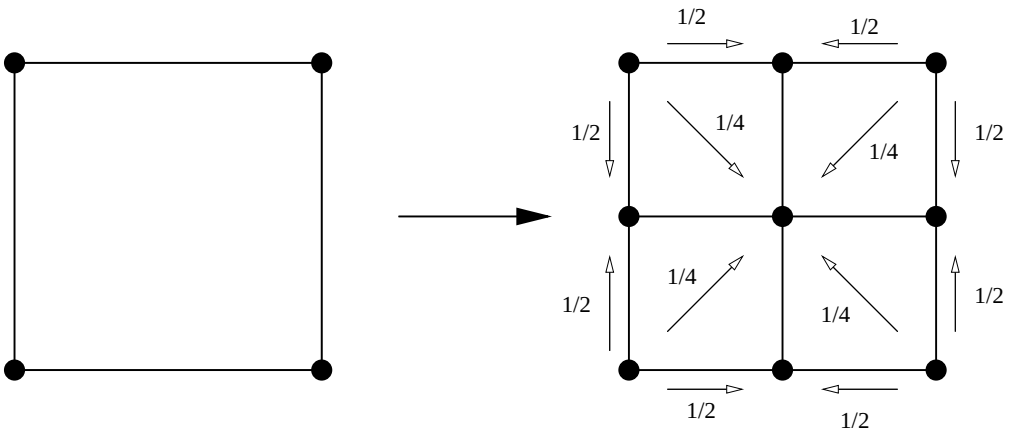


Figure 6.12: Weightings of solution interpolation from coarse to fine grids, for comparisons of solutions produced on different grids.

to do this the two solutions to be compared may be mapped onto a shared uniform grid, using linear interpolation for the points which do not already exist. Figure 6.12 shows the interpolation scheme for the transfer of the solution from a coarse element to its once refined counterpart.

Solutions were found for $t = 0.01$ with locally refined grids of maximum levels 7, 8, 9 and 10, for the backward Euler and trapezoidal discretisations. Differences were calculated between each consecutive pair using the method described above. The L_2-norms of these differences are given in Table 6.5. The difference norm shows clear convergence, reducing by a factor of a little under four as the levels used become finer. Since previous convergence tests, against exact solutions, have shown good results, we can have confidence that this convergence is towards a sufficiently accurate solution. Figure 6.13 shows plots of these differences for the section of the domain containing the highest gradient and hence the most significant errors.

Solution Difference Norm	Backward Euler	Trapezoidal Rule
$\lvert\underline{\omega}^7 - \underline{\omega}^8\rvert_{L_2}$	0.0079664576	0.0078238516
$\lvert\underline{\omega}^8 - \underline{\omega}^9\rvert_{L_2}$	0.0022599768	0.0021833057
$\lvert\underline{\omega}^9 - \underline{\omega}^{10}\rvert_{L_2}$	0.0006518049	0.0006314386

Table 6.5: System of Burgers' equations: L_2-Norm of the difference between solutions found on grids of consecutive maximum grid level.

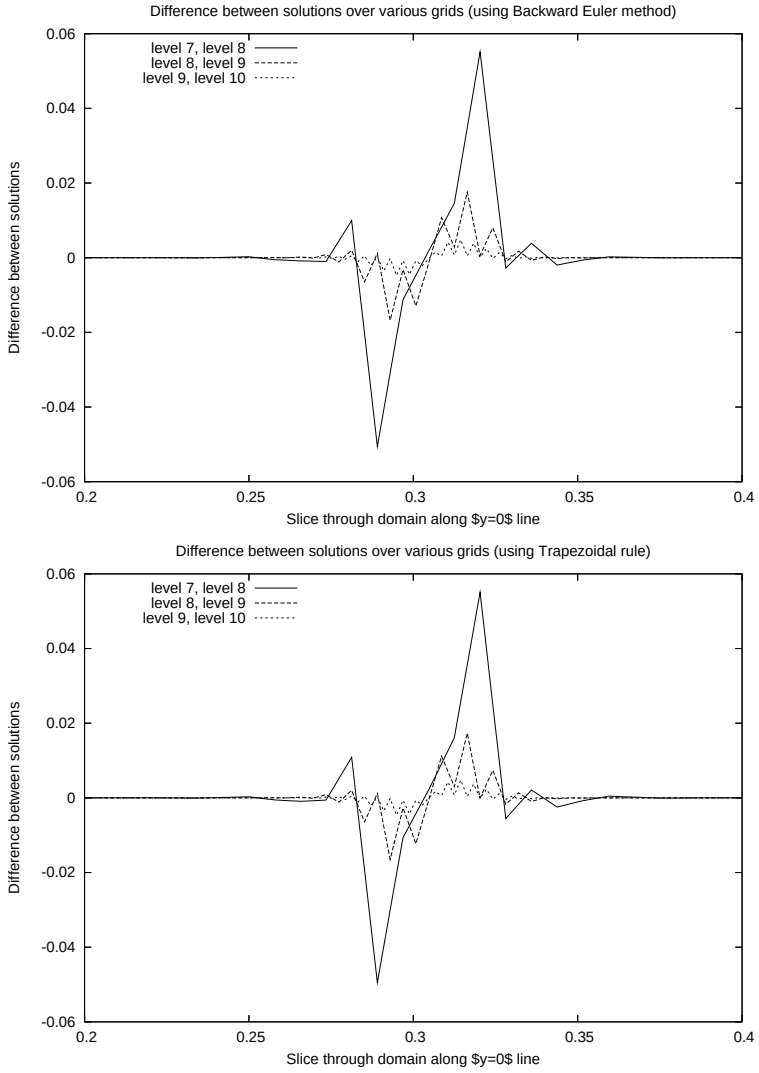


Figure 6.13: System of Burgers' equations: These graphs show the difference between solutions produced on grids with various choices of maximum refinement. A cross section of the domain is taken in each case, along the $y = 0$ line.

Chapter 7

Projected Multigrid for the Implicit Phase Field Model

7.1 Introduction

Having introduced the PFAS algorithm in Chapter 5 and demonstrated its performance on two nonlinear parabolic problems in Chapter 6, in this chapter we apply it to an even more demanding system. A typical example of a phase field (PF) model was introduced in Section 3.4.2 and this was solved adaptively using a semi-implicit time-stepping strategy in Chapter 4. We now show how the PFAS algorithm allows this system to be solved efficiently using both spatial adaptivity and a fully implicit (unconditionally stable) time-step algorithm. The FE discretisation of the anisotropic PF model is shown in the following section. The implicit time discretisation is also undertaken, using the backward Euler method for simplicity. This produces a nonlinear system of equations at each time step. Some details of the implementation of the PFAS solution of this system are given in Sec-

tion 7.3. Finally, we show convergence, accuracy and efficiency results for this complex problem in Section 7.4.

7.2 The Phase Field Model

The governing equations for the PF model, given previously in Chapter 4, that we have used in this work are:

$$\frac{\varepsilon^2}{m}\frac{\partial\phi}{\partial t} = r(\phi,u) + \varepsilon^2\overrightarrow{\nabla}\cdot(s(\phi)\overrightarrow{\nabla}\phi) - \varepsilon^2\frac{\partial}{\partial x}\left(v(\phi)\frac{\partial\phi}{\partial y}\right) + \varepsilon^2\frac{\partial}{\partial y}\left(v(\phi)\frac{\partial\phi}{\partial x}\right), \tag{7.1}$$

$$\frac{\partial u}{\partial t} + \frac{1}{\triangle}P(\phi)\frac{\partial\phi}{\partial t} = \overrightarrow{\nabla}^2 u, \tag{7.2}$$

The parameter values were chosen to be the same as those in Section 4.4.1. The solidification is instigated with a circular 'seed' in the centre of the domain where ϕ takes value of 0 (ϕ has a value of 1 in over most of the domain). This seed, along with a corresponding temperature field, are initialised to the forms $\phi = 1-a$ and $u = -\frac{1}{2}+a$, where

$$a = \frac{1}{1+e^{(r-r_0)/p}}$$

and $r = \sqrt{x^2+y^2}$. We then set r_0 to define the radius of the seed and p to define the steepness of the inter-phase region. For the results given here these are set as: $r_0 = 0.01$ and $p = \varepsilon$ for ϕ and $r_0 = 0.025$ and $p = 8\varepsilon$ for u (to simulate the relative positioning and gentler gradient of the temperature variable). The parameter ε is used as it roughly represents the width of the interface in a typical solution of this problem. Dirichlet conditions were used on the whole of the $(-2,2)\times(-2,2)$ domain.

7.2.1 Spatial Discretisation

In Section 4.4.1 we presented the weak form and FE spatial discretisation of this PF model. The discretised equations (4.13) and (4.14) may be expressed in the following matrix form:

$$\frac{\varepsilon^2}{m}M\dot{\underline{\phi}} = \underline{G}(\underline{\phi},\underline{u}) - \varepsilon^2\underline{K}^w(\underline{\phi}) + \varepsilon^2\underline{H}^{xy}(\underline{\phi}) - \varepsilon^2\underline{H}^{yx}(\underline{\phi}), \tag{7.3}$$

$$M\dot{\underline{u}} = -K\underline{u} - \frac{1}{\triangle}Q(\underline{\phi})\dot{\underline{\phi}}. \tag{7.4}$$

Here M and K are the standard mass and stiffness matrices, introduced in Section 4.4.1. The nonlinear terms are represented as:

$$\underline{G}(\underline{\phi},\underline{u}) = \int_\Omega r(\overline{\phi},\overline{u})N_j d\Omega,$$

$$\underline{K}^w = \varepsilon^2\int_\Omega s(\overline{\phi})(\overrightarrow{\nabla}\overline{\phi}\cdot\overrightarrow{\nabla}N_j)d\Omega,$$

$$\underline{H}^{xy} = \varepsilon^2\int_\Omega \frac{\partial N_j}{\partial x}v(\overline{\phi})\frac{\partial\overline{\phi}}{\partial y}d\Omega,$$

$$\underline{H}^{yx} = \varepsilon^2\int_\Omega \frac{\partial N_j}{\partial y}v(\overline{\phi})\frac{\partial\overline{\phi}}{\partial x}d\Omega,$$

and

$$Q(\underline{\phi}) = \int_\Omega P(\overline{\phi})N_j d\Omega.$$

As before $\overline{u}$ and $\overline{\phi}$ are the discretised functions approximating u and ϕ respectively. Therefore $\overline{u} = \sum_{i=1}^n u_i N_i$ and $\overline{\phi} = \sum_{i=1}^n \phi_i N_i$.

7.2.2 Temporal Discretisation using Backward Euler

The backward Euler method is chosen as the implicit time stepping scheme here since is it a simple method and was shown, in Section 6.3.3, to produce reasonable performance in

conjunction with the PFAS solution scheme in comparison to the higher order trapezoidal rule. (Application of the trapezoidal rule would present no additional difficulties.) In preparation for temporal discretisation, Equations (7.3) and (7.4) can be written as one system in order to isolate the time derivatives:

$$\begin{bmatrix} \frac{\varepsilon^2}{m}M & 0 \\ \frac{1}{\triangle}Q(\underline{\phi}) & M \end{bmatrix} \begin{bmatrix} \dot{\underline{\phi}} \\ \dot{\underline{u}} \end{bmatrix} = \begin{bmatrix} \underline{G}(\underline{\phi},\underline{u}) - \varepsilon^2\underline{K}^w(\underline{\phi}) + \varepsilon^2\underline{H}^{xy}(\underline{\phi}) - \varepsilon^2\underline{H}^{yx}(\underline{\phi}) \\ -K\underline{u} \end{bmatrix}.$$

Since M is non-singular we know that the block triangular matrix is also non-singular and is therefore invertible. This $2n \times 2n$ system can be therefore be rearranged into the form:

$$\begin{bmatrix} \dot{\underline{\phi}} \\ \dot{\underline{u}} \end{bmatrix} = \begin{bmatrix} \frac{\varepsilon^2}{m}M & 0 \\ \frac{1}{\triangle}Q(\underline{\phi}) & M \end{bmatrix}^{-1} \begin{bmatrix} \underline{G}(\underline{\phi},\underline{u}) - \varepsilon^2\underline{K}^w(\underline{\phi}) + \varepsilon^2\underline{H}^{xy}(\underline{\phi}) - \varepsilon^2\underline{H}^{yx}(\underline{\phi}) \\ -K\underline{u} \end{bmatrix}.$$

The backward Euler scheme converts a system of the form $\dot{\underline{x}} = \underline{f}(t,\underline{x})$ to $\underline{x}^{k+1} = \underline{x}^k + \Delta t\,\underline{f}(t^{k+1},\underline{x}^{k+1})$. In this case the discretised form is:

$$\begin{bmatrix} \underline{\phi}^{k+1} \\ \underline{u}^{k+1} \end{bmatrix} = \begin{bmatrix} \underline{\phi}^{k} \\ \underline{u}^{k} \end{bmatrix}$$

$$+\Delta t \begin{bmatrix} \frac{\varepsilon^2}{m}M & 0 \\ \frac{1}{\triangle}Q(\underline{\phi}^{k+1}) & M \end{bmatrix}^{-1} \begin{bmatrix} \underline{G}(\underline{\phi}^{k+1},\underline{u}^{k+1}) - \varepsilon^2\underline{K}^w(\underline{\phi}^{k+1}) + \varepsilon^2\underline{H}^{xy}(\underline{\phi}^{k+1}) - \varepsilon^2\underline{H}^{yx}(\underline{\phi}^{k+1}) \\ -K\underline{u}^{k+1} \end{bmatrix}.$$

To avoid calculating the expensive matrix inversion we rearrange the system again to give:

$$\begin{bmatrix} \frac{\varepsilon^2}{m}M & 0 \\ \frac{1}{\triangle}Q(\underline{\phi}^{k+1}) & M \end{bmatrix} \begin{bmatrix} \underline{\phi}^{k+1} \\ \underline{u}^{k+1} \end{bmatrix} = \begin{bmatrix} \frac{\varepsilon^2}{m}M & 0 \\ \frac{1}{\triangle}Q(\underline{\phi}^{k+1}) & M \end{bmatrix} \begin{bmatrix} \underline{\phi}^{k} \\ \underline{u}^{k} \end{bmatrix}$$

$$+\Delta t \begin{bmatrix} \underline{G}(\underline{\phi}^{k+1}, \underline{u}^{k+1}) - \varepsilon^2 \underline{K}^w(\underline{\phi}^{k+1}) + \varepsilon^2 \underline{H}^{xy}(\underline{\phi}^{k+1}) - \varepsilon^2 \underline{H}^{yx}(\underline{\phi}^{k+1}) \\ -K\underline{u}^{k+1} \end{bmatrix}. \quad (7.5)$$

This algebraic system (7.5) is therefore a the fully discretised representation of this anisotropic PF model.

7.3 Implementation

This section gives some details of the implementation of the PFAS for the algebraic system at each time step. Firstly, we look at the coupling of the two equations. The semi-coupled iterative smoothing method introduced in Section 6.3.2 is also applied here. As in previous implementations described in this thesis the form of the three vectors necessary for calculation of the Projected Jacobi correction are given.

7.3.1 Coupling and Solution

If System (7.5) were to be solved in a fully coupled manner then we could define a vector,

$$\underline{\omega}^{(r)} = \begin{bmatrix} \underline{\phi}^{(r)} \\ \underline{u}^{(r)} \end{bmatrix},$$

where $\underline{\omega}$ is a vector of length $2n$ and the superscript (r) indicates how many smooths have been performed to reach this approximation. The initial conditions are therefore $\underline{\omega}^{(0)}$, the approximation resulting from the first smooth is $\underline{\omega}^{(1)}$, etc. In this case the smoother uses $\underline{\omega}^{(r)}$ to create the new approximation $\underline{\omega}^{(r+1)}$. Both $\underline{\phi}$ and $\underline{u}$ are updated at the same time according to the Jacobi update (described in Section 5.3). We could express this update as:

$$\underline{\omega}^{(r)} \rightarrow \underline{\omega}^{(r+1)}$$

or

$$\begin{bmatrix} \underline{\phi}^{(r)} \\ \underline{u}^{(r)} \end{bmatrix} \rightarrow \begin{bmatrix} \underline{\phi}^{(r+1)} \\ \underline{u}^{(r+1)} \end{bmatrix}.$$

A slightly different approach is taken here however. Each of the approximate solutions to $\underline{\phi}$ and $\underline{u}$ are smoothed independently of each other. We alternate the smooths over the two variables as in a fully coupled method but the approximation of a solution ($\underline{\phi}$ or $\underline{u}$) is updated at the end of its own smooth. The effect of this is a slightly speeded up approximation over the same number of smooths as the fully coupled method. A smooth over $\underline{\phi}$ followed by a smooth over $\underline{u}$, equivalent to one fully coupled smooth, may be expressed as:

$$\begin{bmatrix} \underline{\phi}^{(r)} \\ \underline{u}^{(r)} \end{bmatrix} \rightarrow \begin{bmatrix} \underline{\phi}^{(r+1)} \\ \underline{u}^{(r)} \end{bmatrix} \rightarrow \begin{bmatrix} \underline{\phi}^{(r+1)} \\ \underline{u}^{(r+1)} \end{bmatrix}.$$

The new approximation of the temperature variable, $\underline{u}^{(r+1)}$ has been calculated using the updated approximation of the $\underline{\phi}^{(r+1)}$. The two-stage nature of the semi-coupled smoothing technique gives a slight advantage over a pure Jacobi iteration from this perspective. Therefore algebraic system resulting from each of the original PDEs is smoothed independently of the other. Consequently the form of System (7.5) is not precisely what is solved in this case but an equivalent alternative is used. We extract the components corresponding to each original equation from System (7.5) to form two separate algebraic systems. Extracting the components corresponding to Equation (7.1) out of the system gives:

$$\frac{\varepsilon^2}{m} M \underline{\phi}^{k+1} = \frac{\varepsilon^2}{m} M \underline{\phi}^{k} + \Delta t [\underline{G}(\underline{\phi}^{n+1}, \underline{u}^{k+1}) - \varepsilon^2 \underline{K}^{w}(\underline{\phi}^{k+1}) + \varepsilon^2 \underline{H}^{xy}(\underline{\phi}^{k+1}) - \varepsilon^2 \underline{H}^{yx}(\underline{\phi}^{k+1})],$$

which is rearranged to put all of the terms that contain values at the new time step on the left-hand side:

$$\frac{\varepsilon^2}{m}M\underline{\phi}^{k+1}-\Delta t\underline{G}(\underline{\phi}^{k+1},\underline{u}^{k+1})+\Delta t\varepsilon^2\underline{K}^w(\underline{\phi}^{k+1})-\Delta t\varepsilon^2\underline{H}^{xy}(\underline{\phi}^{k+1})+\Delta t\varepsilon^2\underline{H}^{yx}(\underline{\phi}^{k+1})=\frac{\varepsilon^2}{m}M\underline{\phi}^{k}. \tag{7.6}$$

Similarly, the extracted components of Equation (7.2) may be expressed as:

$$M\underline{u}^{k+1}+\Delta tK\underline{u}^{k+1}+\frac{1}{\triangle}Q(\underline{\phi}^{k+1})[\underline{\phi}^{k+1}-\underline{\phi}^{k}]=M\underline{u}^{k}. \tag{7.7}$$

7.3.2 Jacobi Iteration

In order to discuss the Systems (7.6) and (7.7) we will borrow from the notation of Chapter 6. The following algebraic systems:

$$\hat{\underline{K}}(\underline{\phi}^{k+1},\underline{u}^{k+1})=\hat{\underline{b}},$$

$$\hat{\underline{R}}(\underline{\phi}^{k+1},\underline{u}^{k+1})=\hat{\underline{d}},$$

represent Systems (7.6) and (7.7) respectively, when discretised over a locally refined grid containing hanging nodes. The three vectors that must be constructed in order to form a Projected Jacobi correction to the solution of these equations are $\underline{b}$, $\hat{\underline{K}}(\underline{\phi}^{k+1},\underline{u}^{k+1})$ and $\frac{\partial\hat{K}_j}{\partial\phi_j}(\underline{\phi}^{k+1},\underline{u}^{k+1})$ for System (7.6) and $\underline{d}$, $\hat{\underline{R}}(\underline{\phi}^{k+1},\underline{u}^{k+1})$ and $\frac{\partial\hat{R}_j}{\partial\phi_j}(\underline{\phi}^{k+1},\underline{u}^{k+1})$ for System(7.7). The vectors for the temperature equation system (7.7) will be given first as they are the more straightforward of the two.

Temperature Equation Components

The right-hand side vector, in a point-wise form:

$$\hat{d_j}=\sum_{e=1}^{ne}\left[\int_{\Omega_e}\overline{u}^k N_j d\Omega_e\right],$$

is a simple mass matrix term assembled over each element in the grid. The left-hand side,

$$\hat{R}_j(\underline{\phi}^{k+1},\underline{u}^{k+1}) = \sum_{e=1}^{ne}\left[\int_{\Omega_e}\left\{\overline{u}^{k+1}N_j + \Delta t(\overrightarrow{\nabla}\overline{u}^{k+1}\cdot\overrightarrow{\nabla}N_j) + \frac{1}{\triangle}P(\overline{\phi}^{k+1})[\overline{\phi}^{k+1}-\overline{\phi}^{k}]N_j\right\}d\Omega_e\right],$$

contains only one nonlinear term. We can therefore define the Jacobi vector component-wise as:

$$\begin{aligned}\frac{\partial\hat{R}_j}{\partial u_j}(\underline{\phi}^{k+1},\underline{u}^{k+1}) = \sum_{e=1}^{ne}\Bigg[&\frac{\partial}{\partial u_j}\left\{\int_{\Omega_e}\overline{u}^{k+1}N_j d\Omega_e\right\}\\ &+\frac{\partial}{\partial u_j}\left\{\Delta t\int_{\Omega_e}(\overrightarrow{\nabla}\overline{u}^{k+1}\cdot\overrightarrow{\nabla}N_j)d\Omega_e\right\}\\ &+\frac{\partial}{\partial u_j}\left\{\frac{1}{\triangle}\int_{\Omega_e}P(\overline{\phi}^{k+1})[\overline{\phi}^{k+1}-\overline{\phi}^{k}]N_j d\Omega_e\right\}\Bigg].\end{aligned}$$

The nonlinear term is not dependent on u_j or $\overline{u}$ so this term will be zero in the Jacobian. From the definition of $\overline{u}$ we conclude that $\frac{\partial\overline{u}}{\partial u_j} = N_j$ Therefore the Jacobian vector is simply:

$$\frac{\partial\hat{R}_j}{\partial u_j}(\underline{\phi}^{k+1},\underline{u}^{k+1}) = \sum_{e=1}^{ne}\left[\int_{\Omega_e}\left\{N_j^2 d\Omega + \Delta t(\overrightarrow{\nabla}N_j)^2\right\}d\Omega_e\right].$$

The Phase Equation Components

The right-hand side vector is similar to that of the temperature equation:

$$\hat{b}_j = \sum_{e=1}^{ne}\left[\frac{\varepsilon^2}{m}\int_{\Omega_e}\overline{\phi}^{k}N_j d\Omega_e\right].$$

The left-hand side vector is:

$$\begin{aligned}\hat{K}_j(\underline{\phi}^{k+1},\underline{u}^{k+1}) = \sum_{e=1}^{ne}\Bigg[\int_{\Omega_e}\Bigg\{&\frac{\varepsilon^2}{m}\overline{\phi}^{k+1}N_j - \Delta t r(\overline{\phi}^{k+1},\overline{u}^{k+1})N_j + \varepsilon^2\Delta t s(\overline{\phi}^{k+1})(\overrightarrow{\nabla}\overline{\phi}^{k+1}\cdot\overrightarrow{\nabla}N_j)\\ &+\varepsilon^2\Delta t v(\overline{\phi}^{k+1})\left(\frac{\partial N_j}{\partial y}\frac{\partial\overline{\phi}^{k+1}}{\partial x} - \frac{\partial N_j}{\partial x}\frac{\partial\overline{\phi}^{k+1}}{\partial y}\right)\Bigg\}d\Omega_e\Bigg].\end{aligned}$$

The Jacobian vector is more complicated in this case:

$$\frac{\partial \hat{K}_j}{\partial \phi_j}(\underline{\phi}^{k+1}, \underline{u}^{k+1}) = \sum_{e=1}^{ne} \left[\int_{\Omega_e} \left\{ \frac{\varepsilon^2}{m} N_j^2 - \Delta t \hat{r}_j(\overline{\phi}^{k+1}, \overline{u}^{k+1}) N_j \right.\right.$$

$$+\varepsilon^2 \Delta t \hat{s}_j(\overline{\phi}^{k+1})(\overrightarrow{\nabla}\overline{\phi}^{k+1} \cdot \overrightarrow{\nabla} N_j) + \varepsilon^2 \Delta t s(\overline{\phi}^{k+1})(\overrightarrow{\nabla} N_j)^2$$

$$\left.\left. +\varepsilon^2 \Delta t \hat{v}_j(\overline{\phi}^{k+1}) \left(\frac{\partial N_j}{\partial y} \frac{\partial \overline{\phi}^{k+1}}{\partial x} - \frac{\partial N_j}{\partial x} \frac{\partial \overline{\phi}^{k+1}}{\partial y} \right) \right\} d\Omega_e \right].$$

The functions r, s and v are defined in Equations (4.3), (4.4) and (4.5) respectively. The versions with a hat and a subscript j denote the derivatives of these functions with respect to ϕ_j. For completeness we will give the definition of $\hat{r}_j$, $\hat{s}_j$ and $\hat{v}_j$. From (4.3) we can say that:

$$\hat{r}_j = \frac{\partial r}{\partial \phi_j} = \frac{\partial}{\partial \phi_j} \left(\overline{\phi}(1-\overline{\phi})(\overline{\phi} - 0.5 + a\overline{\phi}(1-\overline{\phi})) \right),$$

where $a = 30\varepsilon\alpha\Delta\overline{u}$. (The superscripts due to the time discretisation have been removed for simplicity.) This evaluates to:

$$\hat{r}_j = -0.5N_j + N_j(3+2a)\overline{\phi} + N_j(-3-6a)\overline{\phi}^2 + 4aN_j\overline{\phi}^3.$$

For the sake for efficiency we will compute this as

$$\hat{r}_j(\overline{\phi}, \overline{u}) = N_j[\overline{\phi}[\overline{\phi}[\overline{\phi}(4a) - (3+6a)] + (3+2a)] - 0.5]$$

In order to calculate $\hat{s}_j$ and $\hat{v}_j$ we require the derivatives of their component parts: η defined in Equation (4.6) and η' defined in Equation (4.7). Both of these functions are dependent on θ. The value of θ is dependent on ϕ and represents the angle between the normal to the solid-liquid interface at a point and the x-axis, it can therefore be calculated as follows:

$$\theta = tan^{-1} \left(\frac{\phi_y}{\phi_x} \right).$$

Since we know that $\frac{\partial}{\partial x} tan^{-1}x = \frac{1}{1+x^2}$ we can write:

$$\hat{\theta}_j = \frac{\partial}{\partial \phi_j}\left[\frac{\phi_y}{\phi_x}\right]\left(\frac{1}{1+(\phi_y/\phi_x)^2}\right) = \left[\frac{\frac{\partial \phi_y}{\partial \phi_j}\phi_x - \phi_y\frac{\partial \phi_x}{\partial \phi_j}}{\phi_x^2}\right]\left(\frac{1}{1+(\phi_y/\phi_x)^2}\right),$$

which can be rearranged to:

$$\hat{\theta}_j = \left[\frac{[N_j]_y\phi_x - \phi_y[N_j]_x}{\phi_x^2+\phi_y^2}\right].$$

(Note that for large sections of the solution these derivatives will be zero. Therefore a safety mechanism must be in place to ensure that the code never divides by zero.) The function $\hat{\eta}_j$ introduces anisotropy into the problem with the formula $\eta(\theta) = 1+\gamma cos(k\theta)$. The derivative, dependent on $\hat{\theta}_j$, is

$$\hat{\eta}_j = \frac{\partial \eta}{\partial \phi_j} = -\gamma k sin(k\theta)\hat{\theta}_j.$$

The second function, dependent on θ is $\hat{\eta'}_j$ which is the derivative of η with respect to θ. Therefore:

$$\hat{\eta'}_j = \frac{\partial \eta'}{\partial \phi_j} = -\gamma k^2 cos(k\theta)\hat{\theta}_j.$$

Now that we have the definition of the components we can give the definition of $\hat{s}_j$ as:

$$\hat{s}_j = \frac{\partial s}{\partial \phi_j} = 2\eta\hat{\eta}_j,$$

and the definition of $\hat{v}_j$ as:

$$\hat{v}_j = \frac{\partial v}{\partial \phi_j} = \hat{\eta}_j\eta' + \eta\hat{\eta'}_j.$$

7.4 Results

In Section 7.4.1 we show the convergence of the PFAS method for the highly nonlinear algebraic system produced by the implicit discretisation above. The cost of the algorithm is re-asserted in Section 7.4.2 where we look at the relationship between the system size and the execution time. Lastly the accuracy of solution is assessed in Section 7.4.3 using the method introduced in Section 6.3.3.

7.4.1 Convergence

A relative tolerance was used to reduce the residual by 10^6 from the initial residual. This was repeated for several levels of finest grid. The initial guess of the solution is zero in order to make the effects of the solver clearer. The results in Figure 7.1 show graphs of the residual against the number of V-cycles. This relationship is shown for the phase and the temperature variables. It is clear that the scheme settles to a uniform rate of convergence on each mesh which does not deteriorate as the mesh level increases.

7.4.2 Efficiency

In Figure 7.2 the graph shows the relationship between system size (the number of nodes in the grid) and overall solution time for a single implicit time step (measured in seconds by a standard C timer). We can see that the time per step is linearly dependent on the system size. The results are taken at regular intervals over a run of 2500 time steps (and the system size is growing because the region of heavy local refinement grows with time). These results reconfirm that our algorithm has a cost of $O(n)$ at each time step.

7.4.3 Accuracy

In the absence of an exact solution, results obtained using various values of finest grid spacing were compared to show that the difference between these solutions converges as

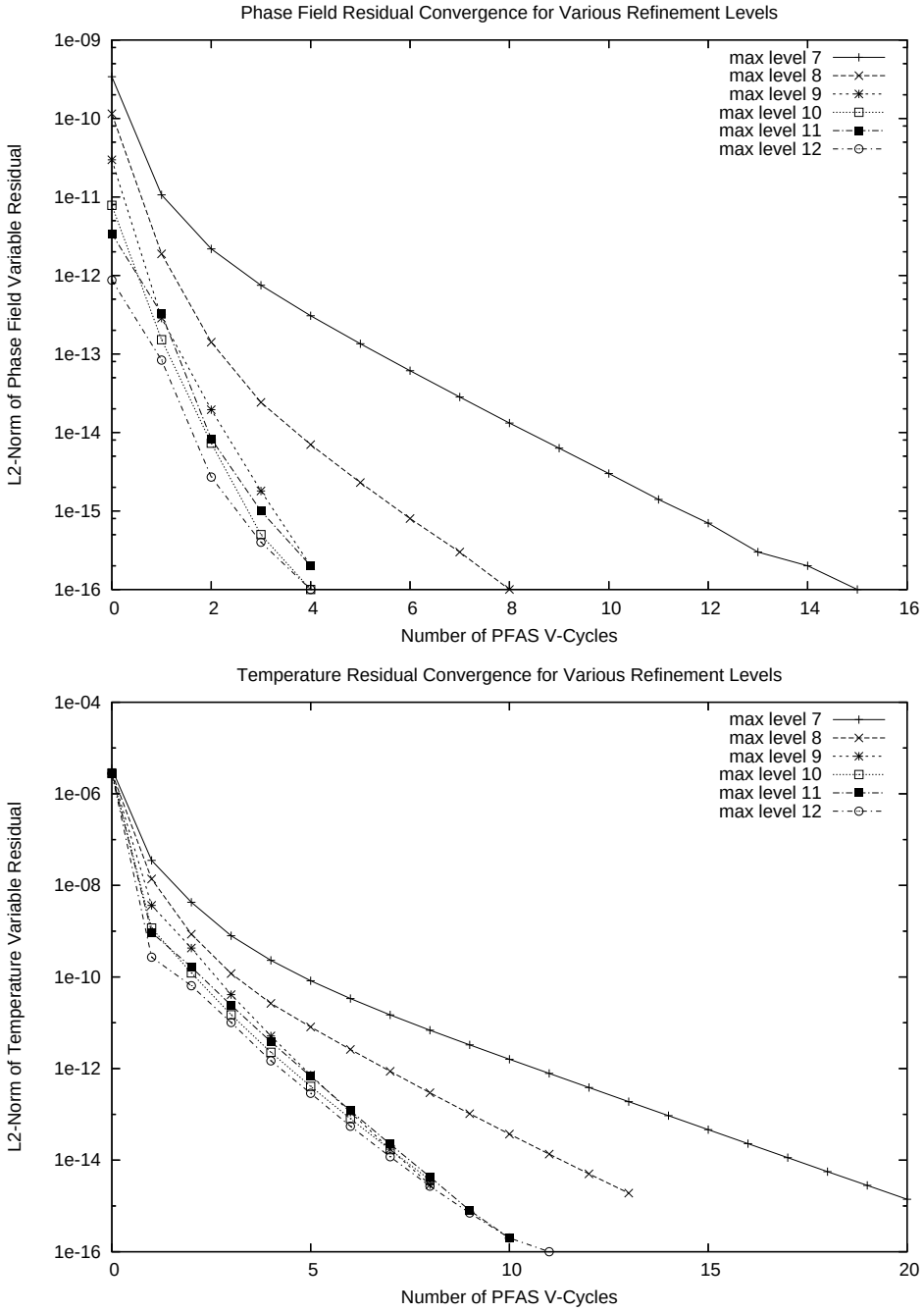


Figure 7.1: Implicit PF model: Convergence of the PFAS: Residual convergence for the ϕ and u variables over a typical implicit time step. The V-cycles were continued until the residual of both approximations met a relative tolerance of 10^{-6}.

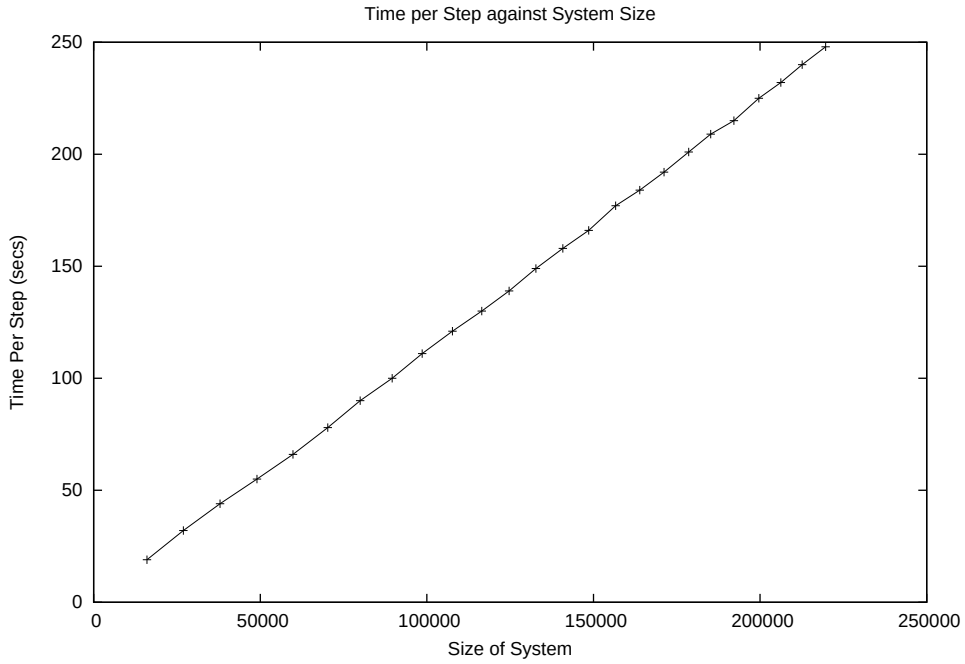


Figure 7.2: Implicit PF model: The execution time for one time step of the PFAS algorithm plotted against the size of the system. These values were found over one solve where the lengthening inter-phase boundary means a steady increase in local refinement.

the finest grid spacing becomes smaller. This method is explained in more detail in Section 6.3.3. Assuming that $\underline{\omega}^{\ell}$ is an approximation of one variable over a grid of maximum refinement level ℓ, the following approximation differences were calculated: $\underline{\omega}^{8} - \underline{\omega}^{9}$, $\underline{\omega}^{9} - \underline{\omega}^{10}$ and $\underline{\omega}^{10} - \underline{\omega}^{11}$, for ϕ and u. The L_2-norm of all of these differences can be seen in Table 7.1. The error average reduced by more than a factor of two with each increase in grid level in the case of ϕ and reduces by a factor of approximately four in the case of u. Figure 7.3 shows graphs of these differences for ϕ and u taken along an x-axis cut through the domain. As the difference between the approximations is significantly higher near the phase change this region has been magnified. The solution convergence with increased local refinement is clearly evident for both of the variables. The differences are more widespread for the temperature variable as the change in this variable happens over a larger section of the domain than that of the phase field variable.

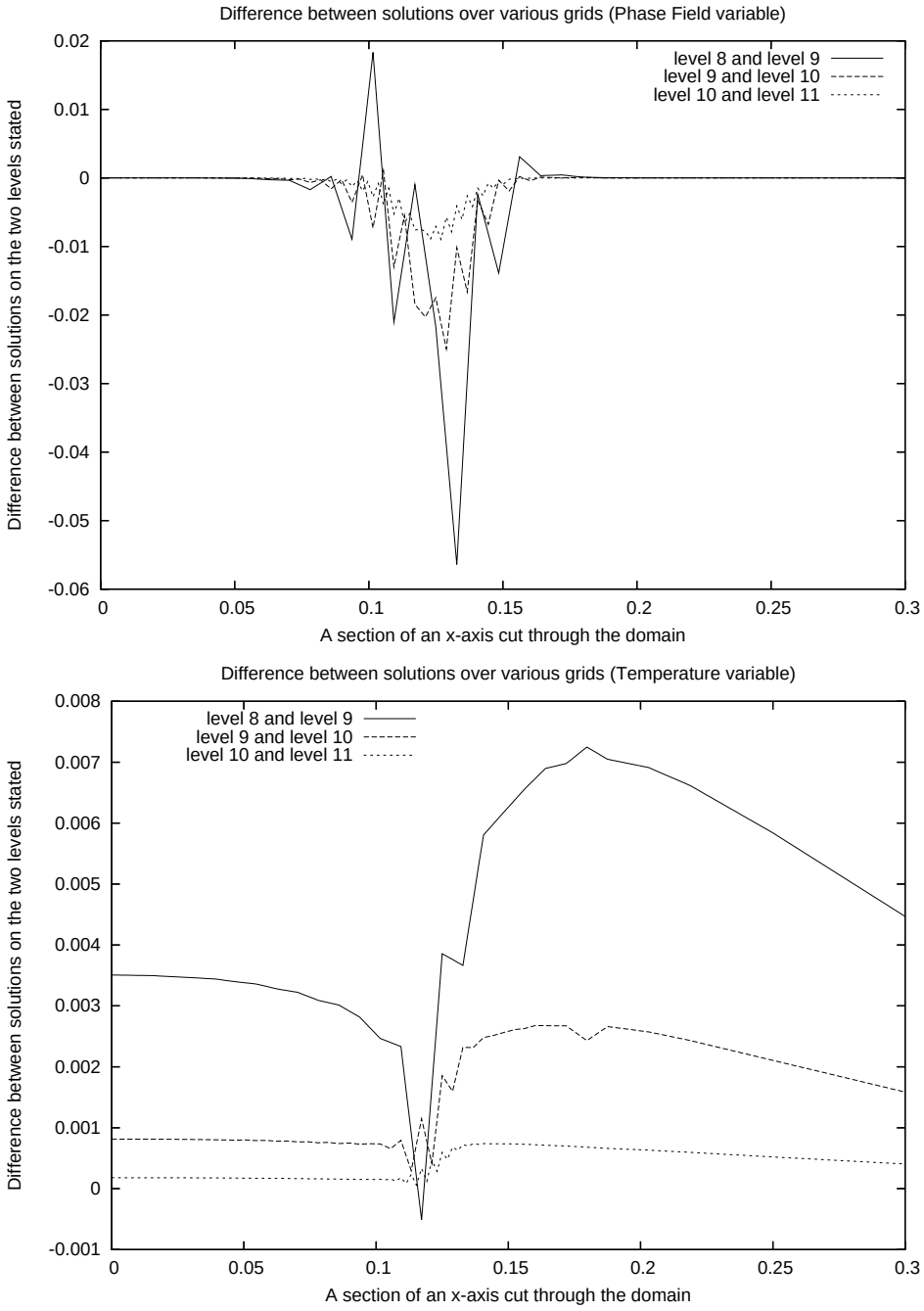


Figure 7.3: Implicit PF model: The first graph shows the differences between approximations to ϕ on various grids and the second shows the same for u. The grids compared have the consecutive finest grid levels stated in the key. These solutions are for $t = 0.04$ with a relative solver tolerance of 10^{-6}.

$\lvert\underline{\omega}^l - \underline{\omega}^{l+1}\rvert_{L_2}$	ϕ	u
$l = 8$	1.0954×10^{-2}	4.4769×10^{-3}
$l = 9$	4.5224×10^{-3}	1.5772×10^{-3}
$l = 10$	1.2674×10^{-3}	3.7724×10^{-4}

Table 7.1: Implicit PF model: Spatial error convergence: The difference in ϕ and the difference in u are shown as an L_2-norm over all of the internal grid points, between various locally refined grids.

7.5 Discussion

In Section 4.4 results are given for the solution of this PF model with a semi-implicit time discretisation. This is an example of an application where aggressive local refinement in a section of the computational domain provides the necessary grid spacing to resolve the solution while keeping the computational time short in comparison to global refinement. In Section 2.5.3 we discussed how allowing hanging nodes in a locally refined mesh reduces the complexity of the meshing algorithms and so this was chosen for use in this work. The PPCG algorithm was used to solve the linear systems resulting from the semi-implicit time discretisation of the PF model in Chapter 4 as it successfully resolves the solution values at the hanging nodes using a projection method. However, in order to use PPCG the algebraic system must be linear. In the case of the PF model a linear algebraic system can only be achieved by using an explicit (or semi-implicit) time stepping scheme. Explicit schemes (discussed in Section 2.2.2) have a stability restriction dependent on the size of the time step. Specifically, the maximum stable time step is linked to the smallest spacing in the grid. As the smallest grid spacing must be small enough to resolve the phase field interface the time step is also restricted to this order (as described in Section 3.4.2). Small time step sizes result in long execution times to produce results of any interest. The PFAS method, developed in this work, overcomes this problem. The PFAS method uses the projection method to resolve the hanging nodes in the locally refined mesh while successfully solving the nonlinear algebraic systems resulting from an implicit discretisation of the PF model. In order to solve nonlinear systems an FAS multi-

Maximum refinement level	Δx	PPCG $Max\|\Delta t\|$	PFAS $Max\|\Delta t\|$
9	0.0156	2.0×10^{-5}	2.0×10^{-4}
10	0.0078	5.0×10^{-6}	5.0×10^{-5}
11	0.0039	1.0×10^{-6}	1.25×10^{-5}
12	0.0019	2.0×10^{-7}	3.125×10^{-6}

Table 7.2: Maximum time step sizes for various maximum grid levels for the PF model solved using the PPCG and the PFAS schemes.

grid method is employed and the smoother (containing the projection steps) is based on a nonlinear Jacobi method. A fully implicit discretisation of the PF model is now a viable option, allowing an increase in the time step sizes and a reduction in the execution time for a solve.

Using the simple first order implicit time stepping scheme we reduce the time step size by a factor of 4 each time the grid spacing is halved to ensure accuracy. Typical time step sizes for the PFAS with this problem are given in the final column of Table 7.2. The corresponding maximum stable time steps for the semi-implicit scheme solved using the PPCG technique are given in the third column. Using the PFAS along with an implicit backward Euler discretisation allows roughly an order of magnitude increase in time step size in comparison to the semi-implicit discretisation and PPCG solution. When using a second order implicit scheme (like the trapezoidal rule) the time step size need only be halved with each increase in the grid refinement level. Therefore, if a second order implicit rule were employed the difference between the explicit and implicit time step sizes would be even greater. Table 7.3 gives time profiling for this implicit phase field implementation using the PFAS solution method. These figures are found from 500 time steps with maximum grid levels of 7, 8 and 9. The settings are chosen to be equivalent to the time profile setting for the semi-implicit implementation in Section 4.4.5. As the FE assembly is repeated within the PFAS iteration it takes a large percentage of the CPU time compared to the PPCG implementation where a linear system is built independent

Maximum refinement level	PFAS Solver	FE Assembly	Adaptive Gridding
7	0.49	98.17	0.13
8	0.63	97.59	0.28
9	0.76	95.51	0.91

Table 7.3: A time profile for the PFAS implementation. Percentages of total execution time are given for the solution process, the finite element assembly and the adaptive gridding.

Maximum refinement level	PPCG Execution time (secs)	PFAS Execution time (secs)
9	22	38
10	232	164
11	3535	1564

Table 7.4: A comparison of the execution time in seconds for the PPCG solution of the semi-implicit phase field model against the PFAS solution of the same phase field model with a fully implicit discretisation.

of the solution process. As would be expected the PFAS solution process (including the FE assembly) is more expensive than the PPCG solution process for a single time step. However, Table 7.4 shows that for fine grids the advantage of taking larger time steps outweighs the expense of the solver, with respect to the CPU time.

Chapter 8

Discussion

In this thesis we have developed a multigrid algorithm for use with adaptively refined grids. A projection technique, for dealing with hanging nodes, originally developed for the Conjugate Gradient method, is applied to a Jacobi iteration. This modified Jacobi is then used as the smoother for a modified FAS multigrid scheme. We have shown that this method is efficient and accurate for nonlinear elliptic problems and nonlinear parabolic problems. This chapter will draw together and review the development and results throughout this thesis. Evaluation of the adaptive gridding techniques and the PFAS solution method is presented in Sections 8.1 and 8.2 respectively. This is followed, in Section 8.3, by a discussion of some possible extensions to the methods proposed.

8.1 Adaptive Gridding Methods

In Chapter 4 we introduced a hierarchical gridding scheme with mechanisms for local refinement and coarsening. The data structures for this scheme are built around a quadtree

structure for the elements in the grid. The re-adaption of the grid uses dynamic memory allocation and de-allocation allowing the grid to change during the execution of the solution process. Due to the structured nature of the grid storage and the imposition of the validity rules (described in Section 4.3.1) our h-adaptive scheme guarantees good quality meshes with at most one hanging node per edge. Examples of such meshes have been illustrated for applications throughout this work, in Figures 4.13, 6.3 and 6.8. The time profiling of implementations in Sections 4.4.5 and 7.5 show that the adaptive gridding procedures do not significantly contribute to CPU time, despite the fact that the grids are re-adapted at regular time steps.

The quadtree data structure successfully facilitates local refinement and coarsening procedures while also naturally accommodating multilevel solution methods. The adaptive gridding implementation in this work strikes a balance between saving memory use and reducing execution time. Clearly, the implementation could be reformed to have a bias towards one or the other. For example, traversing a given set of elements within the quadtree is non-trivial. In this work we choose to implement a set of recursive functions for tree traversal. This task could be accelerated, in some cases, by forming linked lists of the set of elements at each level. This approach is used in [75] for a PF model application.

The data structures and the functionality of the gridding scheme can be extended, in order to facilitate a different application, and indeed they can be cut back, in order to produce an efficient and compact scheme where not all of the functionality is needed. Two possible examples are given below.

1. For some applications it will be unnecessary to store pointers from each node (on each level) to its surrounding elements. The structures holding these pointers (described in Section 5.6.1) are necessary for the multilevel FE build used in conjunction with the PFAS method. However, the PPCG implementation used in Chapter 4,

for example, does not use this functionality.

2. A simple extension of the grid data structures and methods allows its application to finite difference discretisations. Each node (on each level) can be given pointers to its neighbouring nodes allowing for fast access to this information when using the point stencils for discretisation. Indeed this has been successfully undertaken by J. Rosam in order to apply this adaptive meshing software to the finite difference solution of elliptic problems using local refinement.

As the grid structures and methods have been shown to be robust through application to several problems in this work, they can be extended for application to other problems.

8.2 Projected FAS Solution Method

The main novel contribution of this thesis is an adaptive multigrid scheme for the solution of problems defined over locally refined grids containing hanging nodes. This solver has been shown to be effective on nonlinear, elliptic and parabolic problems. In Section 2.5.3 we discussed how allowing hanging nodes to exist within a grid greatly simplifies the refinement and coarsening of certain grid types. The PFAS method allows us to apply multigrid solution method to problems defined on these grids in a relatively simple manner. In Section 8.3.1 the extension to grids other than those with quadrilateral elements is discussed. The way in which the projection steps of the PFAS scheme are formulated allows us to assemble the element matrices in the normal way, without regard for the hanging nodes. The PFAS scheme therefore allows us both to avoid the complication of green refinement and to simplify the resolution of the resulting hanging nodes. Optimal convergence was shown for a nonlinear elliptic problem in Section 5.6.2, for a nonlinear parabolic equation in Section 6.2.5 and for two systems of parabolic equations in Sections 6.3.3 and 7.4. The accuracy of the PFAS method is checked in Sections 5.6.2 and 6.2.5 where exact solutions to the test problems are known. In both cases the discretisation

error is no worse than that obtained using global refinement and it reduces as expected with increased levels of local refinement. The error convergence was verified in the other two problems, where exact solutions were not available, by comparing the solution at various grid levels (Sections 6.3.3 and 7.4). These tests also revealed promising convergence. It has also been shown that the PFAS scheme has a cost which is proportional to the number of points in the grid.

8.3 Possible Extensions

A few of the most logical extensions to the adaptive gridding and solution methods presented in this thesis are discussed in this section. These include the application to different grid types in two and three dimensions (Section 8.3.1), increasing the order of discretisation (Section 8.3.2), applying the method to an even more diverse set of problems (Section 8.3.3) and reviewing the method used to produce the hierarchy of grids for the multigrid solver (Section 8.3.4).

8.3.1 Grid Type

In this work all of the computational grids used contain only square elements. Below is a brief discussion of how the FE grids, and solution methods of this work, could be modified to use triangular grids in two-dimensions and cubic or tetrahedral grids in three-dimensions.

Triangular (2D)

The use of triangular grids has the advantage of increased geometric flexibility in comparison with quadrilaterals. For example, the shape of the domain that we are able to discretise could be an arbitrary polygon. Linear FE basis functions can be used with triangular elements. These are a simpler but less accurate form than the bilinear elements used for

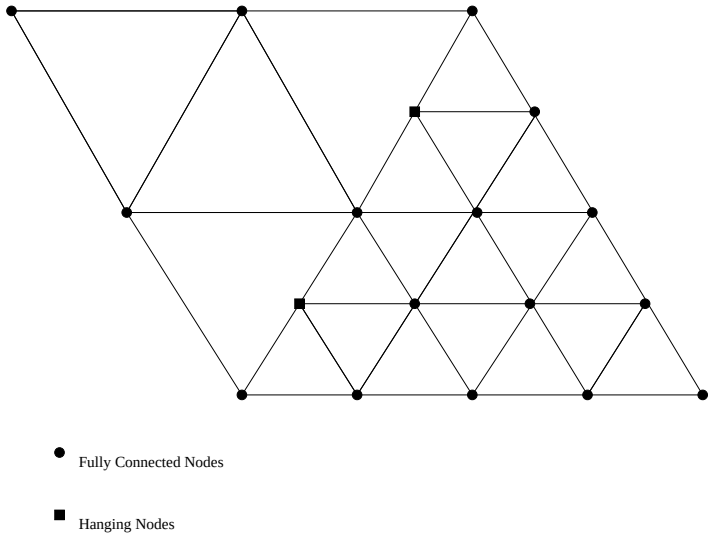


Figure 8.1: Triangular grid refinement: The subdivision of each triangular element into four by creating midpoints at each edge.

our quadrilateral grids. The hierarchical quadtree data structure can still be applied in this case as triangles will naturally divide into four children using the edge midpoints. This is illustrated in Figure 8.1 and is mentioned in Section 2.5.1. Clearly the number of points associated with an element would be three in the case of triangular elements. This reduces the size of the quadtree as each element need only store three pointers for this purpose. However the data structure holding the pointer to all of the elements surrounding each point (on each composite grid level) is more complicated in the triangular case. If all of the triangular elements are equilateral (as in Figure 8.1) then there are six elements surrounding each point. In a less structured grid however this number can not be predetermined. Likewise the system of identifying neighbouring elements of a given element in the grid (described in Section 4.2.3) would have to be revised for the triangular case as the number and orientation of these neighbours is different. From Figure 8.1 we can see that hanging nodes still exist in this grid. Due to the restriction applied to our grids there will still be no more than one hanging node per edge. Therefore the projection methods of the PFAS scheme should be equally effective in this case.

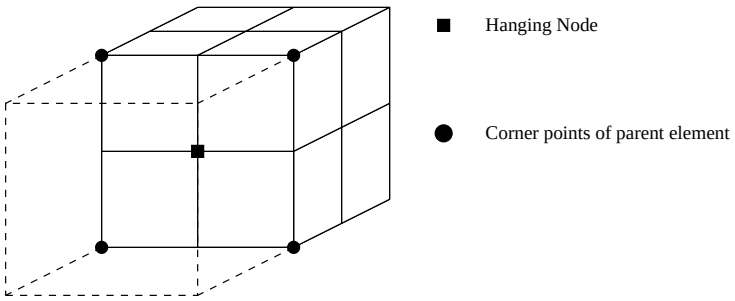


Figure 8.2: Cube element hanging node type A: those which lie in the centre of a parent element's face.

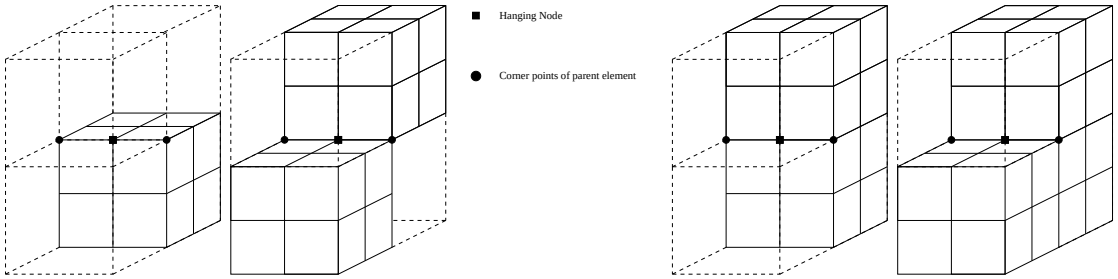


Figure 8.3: Cube element hanging node type B: those which lie in the centre of a parent element's edge.

Cubic (3D)

The extension of the methods presented in this thesis to three dimensions is a natural step. The trilinear FE basis functions can be used for the cubic elements, which are the clear generalisation in a three-dimensional version of this work. The quadtree data structure for the quadrilateral mesh can be extended to an octree structure with eight child elements for each parent element. In this case there are two types of hanging node in a locally refined mesh (referred to here as type A and type B). Those of type A lie on the face of the parent cube. An example of this is given in Figure 8.2. There is only one possible connectivity pattern for this type of hanging node. Hanging nodes of type B lie at the midpoint of an edge of the parent cube and have four possible arrangements of surrounding elements. These possibilities are shown in Figure 8.3 and depend upon the refinement of the neighbouring parent-level cubes. The existence of these two types of hanging node implies the need for two different projection steps within the PFAS scheme. The restric-

tion of hanging nodes of type B should be performed with a projection analogous to the two-dimensional projections, where the values of the two neighbouring points, denoted by circles in Figure 8.3, are used. The restrictions of type A hanging nodes should use an average of four point values. These should be taken from the four points which lie at the four corners of the parent's face containing the hanging node (also denoted by circles in Figure 8.2). In order to correctly align the solution the restriction of type B should be performed before that of type A.

Tetrahedral (3D)

As a tetrahedral element is a simplex for three-dimensional space linear basis functions may be used for the FE assembly. However, the refinement of tetrahedral is not a trivial extension from that of triangles. Grob and Reusken [42] present one method of producing regular tetrahedral refinement. New points are created at the midpoints of each of the tetrahedral edges in a similar manner to regular triangular refinement (shown in Figure 8.1). When these points are joined by edges the tetrahedron is divided into five subsections. Four smaller tetrahedra (of the same shape) are formed in the four corners and an octahedron is formed in the centre. (This is illustrated on the left side of Figure 8.4.) The octahedron can then be divided into three regular tetrahedra. In [87] the hanging nodes are resolved using green refinement as illustrated in the right-hand side of Figure 8.4. This is far more complicated than for triangles in two dimensions since many different green refinement stencils are required for the different cases that may arise. The regular part of the grids (containing hanging nodes) could be used effectively with the PFAS projection method. As there are no face-centred hanging nodes this case is in fact simpler than the cubic case, discussed in the previous section. As the refinement in this method is regular and nested, a tree data structure could again be used to store the elements.

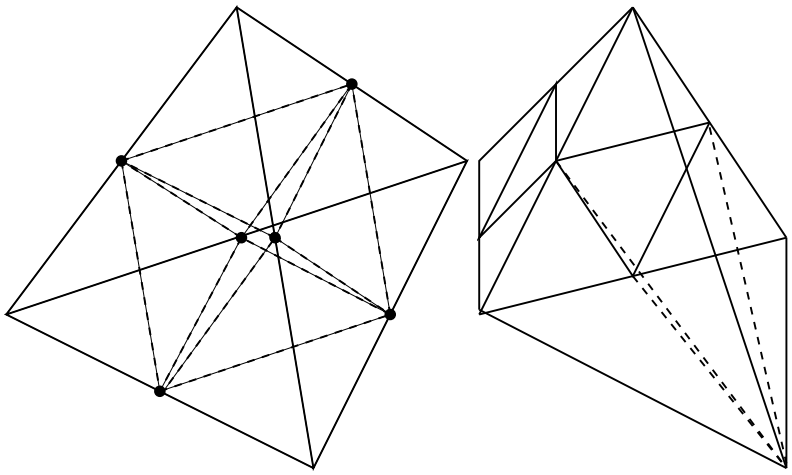


Figure 8.4: Two examples of tetrahedral refinement: The picture on the left shows regular refinement and the one on the right shows the resolution of hanging nodes by means of 'green' refinement.

8.3.2 Discretisation

The FE discretisation in this work used only bilinear basis functions and the time discretisation took simple uniform steps. These methods are sufficient to show the performance of the PFAS solver. However, this section briefly discusses the possibility of enriching these two areas to create a more sophisticated approximation scheme.

Higher Order Spatial Discretisation

Within the Galerkin FE method we can increase the order of the discretisation by increasing the order of the basis functions over each element. Until now we have only considered linear and bilinear basis functions, here we take a brief look at the quadratic version. Quadratic bases can be used for triangular elements in two dimensions and tetrahedral elements in three dimensions. In order to establish a quadratic function over a single element, extra interpolation points are required. (These points increase the size of the system making the solution more expensive.) For quadratics the extra interpolation points are generally chosen as the midpoints of each of the elements edges. This is shown for the triangular and tetrahedral cases, in Figure 8.5.

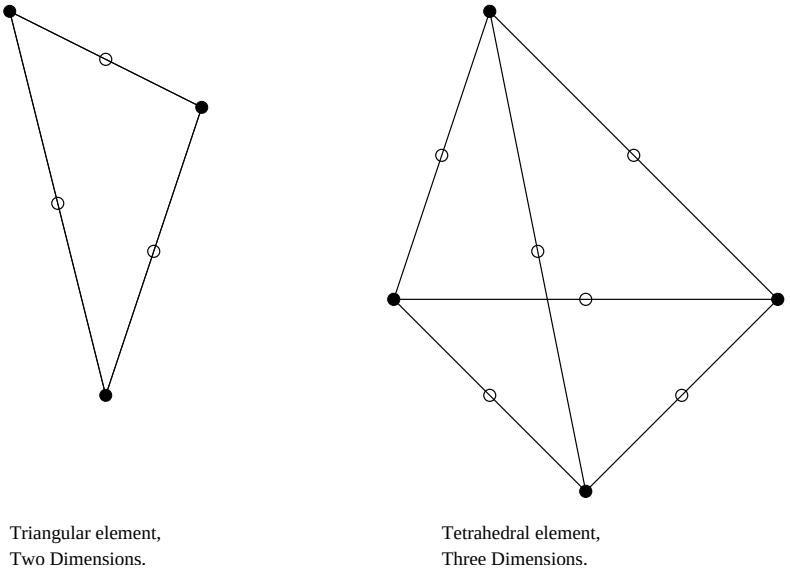


Figure 8.5: Interpolation points for quadratic finite elements for triangular and tetrahedral elements in two and three dimensions respectively.

For the quadrilateral grids, used throughout this work, bi-quadratic functions can be used. In this case an interpolation point is needed at the centre of the element as well as at the midpoint of each edge. This is illustrated in Figure 8.6 for a segment of non-uniform mesh. In order to apply bi-quadratic finite elements within the Projected FAS scheme we have to analyse what happens along edges such as edge E in Figure 8.6 where there exist hanging nodes. Another representation of the edge, E, is given in Figure 8.7. In this

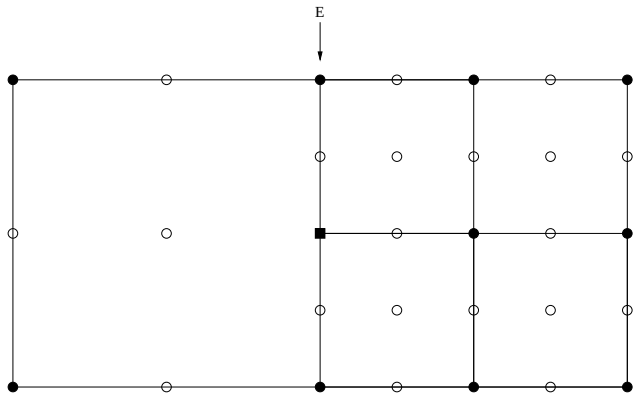


Figure 8.6: Interpolation points for bi-quadratic finite element: Shown here for a section of a non-uniform, two-dimensional grid.

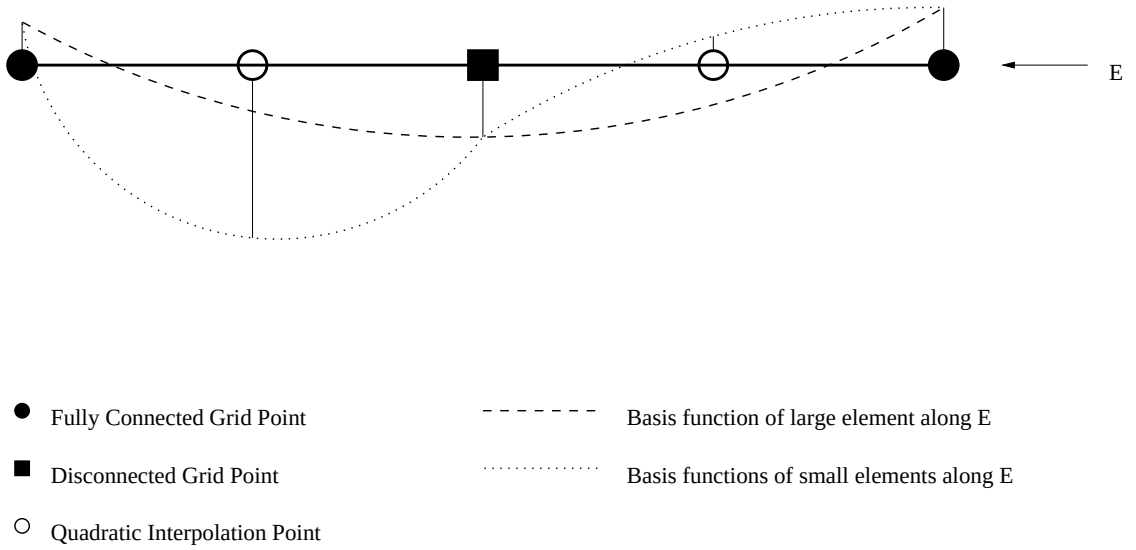


Figure 8.7: Hanging nodes in a non-uniform, bi-quadratic FE grid: A representation of the discontinuity in the solution created along edges containing hanging nodes for bi-quadratic finite elements.

figure there are arbitrary values given for each point and a representation of the quadratic interpolation functions are given for the large element and for the smaller elements. This shows clearly that, as in the linear case, a discontinuity in the solution would generally be caused along this edge. A projection method would therefore have to be formulated in order to force the quadratic functions of the smaller elements to match that of the larger element along this edge. In this bi-quadratic case the value at the two hanging nodes (denoted ◯) should be the same as the value at these points on the large element. These values should therefore be prescribed by the projection step whenever it is required as part of the PFAS procedure.

Bi-quadratic basis functions can be extended to be used on three-dimensional cubic grids (termed tri-quadratics). Although working in three dimensions adds a level of complexity to the PFAS projection methods (as discussed in Section 8.3.1) the principles are the same.

Adaptive Time Discretisation

Adaptivity can be employed in time as well as in space. The time step size Δt can be decided for each individual time step. Allowing variable time step sizes can often produce the final solution of an initial value problem in fewer steps. However, even when implicit time stepping schemes are used, we must restrict the time step size in order to maintain the desired accuracy. The time step sizes are therefore chosen based on some local error estimation, le. At each step Δt is chosen to be as large as possible without exceeding some tolerance, $ttol$. One example of a method for estimating the local error is to compare the solution of one time step found using a method of order p with the solution of the same step found using a method of order $p+1$. The difference between these solutions is used as an estimate to the local error and, from this, an estimate can be made to the optimal time step. An example of this type of method is the Runge-Kutta-Fehlberg method [22] where two Runge-Kutta schemes of consecutive order are used to form the local error estimate. If $le < tol$ the higher order solution for this time step is used and the next step is taken using the optimal Δt. However, if $tol < le$ then this time step is re-calculated using the optimal Δt. These variable time stepping methods would be an addition to the flexibility of the solution methods which would not interfere with the existing adaptive gridding and PFAS methods for the boundary-value problems at each step.

8.3.3 Applications

The most natural extension, in terms of applications, would seem to be to the class of hyperbolic PDEs. Hyperbolic equations are often used to simulate flows and other transport problems when diffusive effects may be neglected (for example the Euler equations [71]). Galerkin FE methods are typically unstable in the approximate solution for this class of problems, usually leading to highly oscillatory numerical solutions. In order to resolve this, 'upwinding' discretisation methods have been developed. One example of this is the Streamline Upwind Petrov Galerkin (SUPG) scheme, developed by Brooks *et al* [20]. The

finite element test functions are enriched to form an upwinded version of the Galerkin FE scheme. Extending our PFAS method to accommodate such stabilisation depends mainly on being able to develop a multigrid solver that works efficiently on a uniform sequence of grids. In this case the simplicity of the PFAS method for locally refined grids should still hold. An alternative, such as the discontinuous Galerkin method should also be considered. It should also be noted that there are existing adaptive multigrid schemes which have been successfully applied to hyperbolic problems, such as that of Berger [1].

8.3.4 Multigrid Coarsening

There is room for improvement in the method by which the non-uniform grids, in the PFAS scheme, are coarsened to create the multigrid hierarchy of grids. Currently the grids are coarsened by simply removing all elements at the finest level in the grid. This is, of course, a fast and cheap coarsening method since it reflects the order in which the grids were created by the local refinement process. A, more sophisticated, alternative is given by Aftosmis *et al* in [1]. Here the grid is coarsened by one refinement level wherever possible over the whole grid. Figure 8.8 gives three of the grids from a hierarchy created with the method given in Chapter 4 and three equivalent grids from a hierarchy created using the ideas in [1]. (The exact form of the grids in the proposed scheme take into consideration the rules of de-refinement defined in Chapter 4.) The proposed scheme is straightforward and a viable alternative to our current methods. It can be noted that the PFAS scheme is optimal with both of these multigrid coarsening schemes so we would expect a relatively small advantage from using the approach of [1].

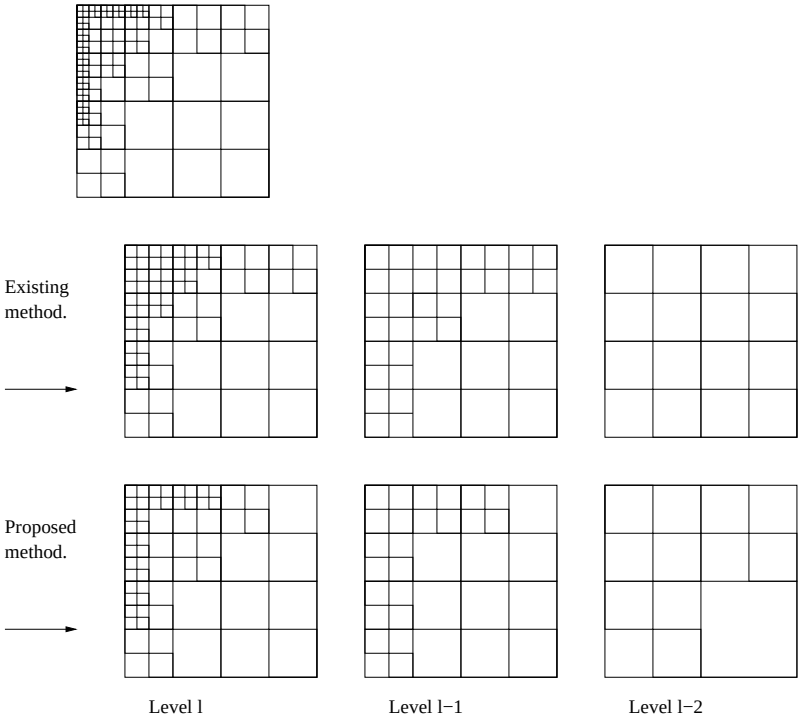


Figure 8.8: An alternative coarsening method for creating multigrid levels.

Bibliography

[1] Aftosmis M.J., Berger M.J. and Adomavicius G. 'A parallel multilevel method for adaptively refined cartesian grids with embedded boundaries.' *AIAA. 38th Aerospace Sciences Meeting 2000-0808*, 2000.

[2] Ainsworth M. and Oden J.T. *A posterior error estimation in finite element analysis*. Wiley, 2000.

[3] ANSYS. 'The CFX homepage.', last accessed: 25/09/05. `http://www-waterloo.ansys.com/cfx/`.

[4] Arnold C.B., Aziz M.J., Schwarz M. and Herlach D. 'Parameter-free test of alloy dendrite-growth theory.' *Physical Review B*, Vol 59(1), pp. 334–343, 1999.

[5] Arnold D.N., Mukherjee A. and Pouly L. 'Locally adapted tetrahedral meshes using bisection.' *SIAM Journal on Scientific Computing*, Vol 22(2), pp. 431–448, 2000.

[6] Axelsson O. *Iterative Solution Methods*. Cambridge University Press, 1996.

[7] Babuska I. and Rheinboldt W.C. 'A posteriori error analysis of finite element solutions for one dimensional problems.' *SIAM Journal on Numerical Analysis*, Vol 18, pp. 565–589, 1981.

[8] 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*, Vol 54, pp. 450–469, 2005.

[9] Bank R.E. *PLTMG: A software package for solving elliptic partial differential equations. User's Guide 6.0*. SIAM, Philadelphia, 1990.

[10] Bank R.E. and Weiser A. 'Some a posteriori error estimates for elliptic partial differential equations.' *Mathematics of Computation*, Vol 44, pp. 283–301, 1985.

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

[12] Bates P.W., Fife P.C., Gardner R.A. and Jones C.K.R.T. 'Phase-field models for hypercooled solidification.' *Physica D*, Vol 104(1), pp. 1–31, 1997.

[13] Bell J., Berger M., Saltzman J. and Welcome M. 'Three-dimensional adaptive mesh refinement for hyperbolic conservation laws.' *SIAM Journal on Scientific Computing*, Vol 15(1), pp. 127–138, 1994.

[14] Benito J., Urena F., Gavete L. and Alvarez R. 'An h-adaptive method in the generalized finite differences.' *Computational Methods for Applied Mechanical Engineering*, Vol 192, pp. 735–759, 2003.

[15] Berger M.J. and Colella P. 'Local adaptive mesh refinement for shock hydrodynamics.' *Journal of Computational Physics*, Vol 82, pp. 64–84, 1989.

[16] Berger M.J. and Oliger J. 'Adaptive mesh refinement for hyperbolic partial differential equations.' *Journal of Computational Physics*, Vol 53, pp. 484–512, 1984.

[17] Berzins M. and Furzeland R.M. 'An adaptive theta method for the solution of stiff and non-stiff differential equations.' *Applied Numerical Mathematics*, Vol 9, pp. 1–19, 1992.

[18] Brandt A. 'Multi-level adaptive solutions to boundary-value problems.' *Mathematics of Computation*, Vol 31(138), pp. 333–390, 1977.

[19] Briggs W.L., Henson V.E. and McCormick S.F. *A Multigrid Tutorial*. SIAM, 2000.

[20] Brooks A.N. and Hughes T.J.R. 'Streamline Upwind / Petrov-Galerkin formulation for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations.' *Computer Methods in Applied Mechanics and Engineering*, Vol 32, pp. 199–259, 1982.

[21] Browne D.J. and Hunt J.D. 'A fixed grid front-tracking model of the growth of a columnar front and an equiaxed grain during solidification of an alloy.' *Numerical Heat Transfer Part B-Fundamentals*, Vol 45(5), pp. 395–419, 2004.

[22] Burden R.L. and Faires J.D. *Numerical Analysis*. Wadsworth Group, Brooks/Cole, 2001.

[23] Burgers J.M. *The Nonlinear Diffusion Equation*. D. Reidel Publishing Company, 1974.

[24] Butrylo B., Vollaire C., Nicolas L. and Nicolas A. 'Numerical performance of the distributed vector finite-element time-domain algorithm.' *IEEE Transactions on Magnetics*, Vol 40(2), pp. 997–1000, 2004.

[25] Chang R.Y. and Hsu C.H. 'A variable-order spectral element method for incompressible viscous flow simulation.' *International Journal for Numerical Methods in Engineering*, Vol 39(17), pp. 2865–2887, 1996.

[26] Coorevits P. and Bellenger E. 'Alternative mesh optimality criteria for h-adaptive finite element method.' *Finite Elements in Analysis and Design*, Vol 40, pp. 1195–1215, 2004.

[27] Cramer H., Rudolph M., Steinl G. and Wunderlich W. 'A hierarchical adaptive finite element strategy for elastic-plastic problems.' *Computers and Structures*, Vol 73, pp. 61–72, 1999.

[28] Daskalopoulos P. and Hamilton R. 'Regularity of the free boundary for the porous medium equation.' *Journal of the Mathematical Society*, Vol 11(4), pp. 895–965, 1998.

[29] Daskalopoulos P. and Rhee E. 'Free-boundary regularity for generalized porous medium equations.' *Communications on Pure and Applied Analysis*, Vol 2(4), pp. 481–494, 2003.

[30] 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*, Vol 3(1), pp. 6–27, 1982.

[31] Dawson C. and Aizinger V. 'A discontinuous Galerkin method for three-dimensional shallow water equations.' *Journal of Scientific Computing*, Vol 22(1), pp. 245–267, 2004.

[32] Ehlers W., Ammann M. and Diebels S. 'h-adaptive FE methods applied to single- and multiphase problems.' *International Journal for Numerical Methods in Engineering*, Vol 54, pp. 219–239, 2002.

[33] Ekevid T., Kettil P. and Wiberg N.E. 'Adaptive multigrid for finite element computations in plasticity.' *Computers and Structures*, Vol 82(28), pp. 2413–2424, 2004.

[34] Esakov J. and Weiss T. *Data Structures: An Advanced Approach Using C*. Prentice-Hall, 1989.

[35] Ewing R.E. and Lazarov R.D. 'Approximation of parabolic problems on grids locally refined in time and space.' *Applied Numerical Mathematics*, Vol 14, pp. 199–211, 1994.

[36] Fidkowski K.J., Oliver T.A., Lu J. and Darmofal D.L. 'p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations.' *Journal of Computational Physics*, Vol 207, pp. 92–113, 2005.

[37] Fluent. 'The Fluent homepage.', last accessed: 25/09/05. `http://www.fluent.com`.

[38] Gaskell P.H., Jimack P.K., Sellier M. and Thompson H.M. 'Efficient and accurate time adaptive multigrid simulations of droplet spreading.' *International Journal for Numerical Methods in Fluids*, Vol 45(11), pp. 1161–1186, 2004.

[39] Gentleman W.M. and George A. 'Sparse matrix software.' In: J. Bunch and D. Rose (editors), *Sparse Matrix Computations*, pp. 243–261. Academic Press, New York, 1976.

[40] Golub G.E. and Loan C.F.V. *Matrix Computations*. The Johns Hopkins University Press, 1989.

[41] Gottlieb D. and Gottlieb S. 'Spectral methods for compressible reactive flows.' *Comptes Rendus Mechanique*, Vol 333(1), pp. 3–16, 2005.

[42] Grob S. and Reusken A. 'Parallel multilevel tetrahedral grid refinement.' *SIAM Journal of Scientific Computation*, Vol 26(4), pp. 1261–1288, 2005.

[43] Gupta A.K. 'A finite element transition from a fine to a coarse grid.' *International Journal for Numerical Methods in Engineering*, Vol 12, pp. 35–45, 1978.

[44] Hart L., McCormick S. and O'Gallagher A. 'The fast adaptive composite-grid method algorithms for advanced computers.' *Applied Mathematics and Computation*, Vol 19, pp. 103–125, 1986.

[45] Huang W. and Russell R.D. 'A high dimensional moving mesh strategy.' *Applied Numerical Mathematics*, Vol 26, pp. 63–76, 1998.

[46] Hyman J.M., Knapp R.J. and Scovel J.C. 'High order finite volume approximations of differential operators on nonuniform grids.' *Physica D*, Vol 60, pp. 112–138, 1992.

[47] Jeong J.H., Goldenfeld N. and Dantzig J.A. 'Phase-field model for three-dimensional dendritic growth with fluid flow.' *Physical Review E*, Vol 64, pp. 041602 (1–14), 2001.

[48] Jouhaud J.C., Montagnac M. and Tourrette L. 'A multigrid adaptive mesh refinement strategy for 3d aerodynamic design.' *International Journal for Numerical Methods in Fluids*, Vol 47(5), pp. 367–385, 2004.

[49] Karma A., Lee Y.H. and Plapp M. 'Three-dimensional dendrite-tip morphology at low undercooling.' *Physical Review E*, Vol 61(4), pp. 3996–4006, 2000.

[50] Karma A. and Rappel W. 'Phase-field method for computationally efficient modeling of solidification with arbitrary interface kinetics.' *Physical Review E*, Vol 53(4), pp. 3017–3020, 1996.

[51] Karma A. and Rappel W. 'Phase-field simulation of three-dimensional dendrites: is microscopic solvability theory correct?' *Journal of Crystal Growth*, Vol 174, pp. 54–64, 1997.

[52] Karma A. and Rappel W. 'Quantitative phase field modeling of dendritic growth in two and three dimensions.' *Physical Review E*, Vol 57(4), pp. 4323–4349, 1998.

[53] Kelly D.W., Gago J.R., Zienkiewicz O.C. and Babuska I. 'A posteriori error analysis and adaptive processes in the finite element method. part i–error analysis.' *International Journal for Numerical Methods in Engineering*, Vol 19, pp. 1593–1619, 1983.

[54] Knoll D.A. and McHugh P.R. 'Newton-Krylov methods applied to a system of convection-diffusion-reaction equations.' *Computer Physics Communications*, Vol 88, pp. 141–160, 1995.

[55] Kobayashi R. 'Modeling and numerical simulations of dendritic crystal growth.' *Physica D*, Vol 63, pp. 410–423, 1993.

[56] Kouatchou J. 'Comparison of time and spatial collocation methods for the heat equation.' *Journal of Computational and Applied Mathematics*, Vol 150(1), pp. 129–141, 2003.

[57] Kozlovsky G. 'Solving partial differential equations using recursive grids.' *Applied Numerical Mathematics*, Vol 14, pp. 165–181, 1994.

[58] Li X.Y.S. and Demmel J.W. 'SuperLU DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems.' *ACM Transactions on Mathematical Software*, Vol 29(2), pp. 110–140, 2003.

[59] Lohner R. 'An adaptive finite elememt scheme for transient problems in CFD.' *Computer Methods in Applied Mechanics and Engineering*, Vol 61, pp. 323–338, 1987.

[60] Lohner R. 'Adaptive remeshing for transient problems.' *Computer Methods in Applied Mechanics and Engineering*, Vol 75, pp. 195–214, 1989.

[61] Madden N. and Stynes M. 'Efficient generation of oriented meshes for solving convection-diffusion problems.' *International Journal for Numerical Methods in Engineering*, Vol 40, pp. 565–576, 1997.

[62] Mavriplis D.J. 'Unstructured grid techniques.' *Annual Review of Fluid Mechanics*, Vol 29, pp. 473–514, 1997.

[63] Mavriplis D.J. 'Multigrid approaches to non-linear diffusion problems on unstructured meshes.' *Numerical Linear Algebra with Applications*, Vol 8(1), pp. 499–512, 2001.

[64] McCormick S. and Rude U. 'A finite volume convergence theory for the fast adaptive composite grid methods.' *Applied Numerical Mathematics*, Vol 14(0), pp. 91–103, 1994.

[65] McCormick S. and Thomas J. 'The fast adaptive composite grid (FAC) method for elliptic equations.' *Mathematics of Computation*, Vol 46(174), pp. 439–456, 1986.

[66] McCormick S.F. *Multilevel adaptive methods for partial differential equations.*. SIAM, 1989.

[67] Meyer A. 'Projection technique embedded in the PCGM for handling hanging nodes and boundary restrictions.' In: B. Topping and Z. Bittnar (editors), *Engineering Computational Technology*, pp. 147–165. Saxe-Coburg, Stirling, Scotland, 2002.

[68] Miller K. and Baines M.J. 'Least squares moving finite elements.' *IMA Journal of Numerical Analysis*, Vol 21(3), pp. 621–642, 2001.

[69] Mullis A.M. 'A study of kinetically limited dendritic growth at high undercooling using phase-field techniques.' *Acta Materialia*, Vol 51, pp. 1959–1969, 2003.

[70] Oden J.T., Babuska I. and Baumann C.E. 'A discontinuous hp finite element method for diffusion problems.' *Journal of Computational Physics*, Vol 146, pp. 491–519, 1998.

[71] Oden J.T., Strouboulis T. and Devloo P. 'Adaptive finite element methods for the analysis of inviscid compressible flow: Part 1. fast refinement / unrefinement and moving mesh methods for unstructured meshes.' *Computer Methods in Applied Mechanics and Engineering*, Vol 59, pp. 327–362, 1986.

[72] Ortega J.M. and Rheinbolt W.C. *Iterative Solution of Non-linear Equations in Several Variables*. Academic Press, 1970.

[73] Pennington S.V., Jimack P.K. and McFarlane K. 'Adaptive numerical simulation of the remediation of soil contamination in in-situ gas venting.' *Computational Geosciences*, Vol 3(2), pp. 135–160, 1999.

[74] Picasso M. 'An anisotropic error indicator based on Zienkiewicz-Zhu error estimator: application to elliptic and parabolic problems.' *SIAM Journal on Scientific Computing*, Vol 24(4), pp. 1328–1355, 2003.

[75] Provatas N., Goldenfeld N. and Dantzig J. 'Adaptive mesh refinement computation of solidification microstructure using dynamic data structures.' *Journal of Computational Physics*, Vol 148, pp. 265–290, 1999.

[76] Reddy J.N. *Finite Element Method*. McGraw-Hill, 1993.

[77] Rheinboldt W.C. and Mesztenyi C.K. 'On a data structure for adaptive finite element mesh refinements.' *ACM Transactions on Mathematical Software*, Vol 6(2), pp. 166–187, 1980.

[78] Richtmyer R.D. and Morton K.W. *Difference Methods for Initial-Value Problems*. Interscience Publishers (division of John Wiley & Sons), 1967.

[79] Rivara M.C. 'Design and data structure of fully adaptive, multigrid, finite element software.' *ACM Transactions on Mathematical Software*, Vol 10(3), pp. 242–264, 1984.

[80] Rivara M.C. 'A grid generator based on 4-triangles conforming mesh-refinement algorithms.' *International Journal for Numerical Methods in Engineering*, Vol 24, pp. 1343–1354, 1987.

[81] Rude U. 'Fully adaptive multigrid methods.' *SIAM Journal on Numerical Analysis*, Vol 30(1), pp. 230–248, 1993.

[82] Rude U. 'On the multilevel adaptive iterative method.' *SIAM Journal on Scientific Computation*, Vol 15(3), pp. 577–586, 1994.

[83] Saad Y. and Schultz M. 'GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems.' *SIAM Journal on Scientific and Statistical Computing*, Vol 7, pp. 856–869, 1986.

[84] Saad Y. and van der Vorst H.A. 'Iterative solution of linear system in the 20-th century.' *Journal of Computational and Applied Mathematics*, Vol 123, pp. 35–65, 2000.

[85] Schmidt A. 'Computation of three dimensional dendrites with finite elements.' *Journal of Computational Physics*, Vol 125(0095), pp. 293–312, 1996.

[86] Smith G.D. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford University Press, 1985.

[87] Spears W. and Berzins M. 'A 3D unstructured mesh adaptation algorithm for time-dependent shock-dominated problems.' *International Journal for Numerical Methods in Fluids*, Vol 25(1), pp. 81–104, 1997.

[88] Strain J. 'Tree methods for moving interfaces.' *Journal of Computational Physics*, Vol 151, pp. 616–648, 1999.

[89] Tabarraei A. and Sukumar N. 'Adaptive computations on conforming quadtree meshes.' *Finite Elements in Analysis and Design*, Vol 41, pp. 686–702, 2005.

[90] Teman R. *Navier Stokes Equations: theory and numerical analysis*. Oxford University Press, 2001.

[91] Thomee V. 'From finite differences to finite elements. A short history of numerical analysis of partial differential equations.' *Journal of Computational and Applied Mathematics*, Vol 128, pp. 1–54, 2001.

[92] Trottenberg U., Oosterlee C. and Schuller A. *Multigrid*. Academic Press, London, 2001.

[93] Versteeg H.K. and Malalasekera W. *An Introduction to Computational Fluid Dymanics. The Finite Volume Method.*. Longman, 1995.

[94] Walters R.A. and Barragy E.J. 'Comparison of h and p finite element approximations of the shallow water equations.' *International Journal for Numerical Methods in Fluids*, Vol 24(1), pp. 61–79, 1997.

[95] Wang W. 'Special bilinear quadrilateral elements for locally refined finite element grids.' *SIAM Journal on Scientific Computing*, Vol 22, pp. 2029–2050, 2001.

[96] Wathen A.J. 'Realistic eigenvalue bounds for the Galerkin mass matrix.' *IMA Journal of Numerical Analysis*, Vol 7, pp. 449–457, 1987.

[97] Wheeler A.A., Murray B.T. and Schaefer R.J. 'Computation of dendrites using a phase field model.' *Physica D*, Vol 66, pp. 243–262, 1993.

[98] Zave P. and Rheinboldt W.C. 'Design of an adaptive, parallel finite-element system.' *ACM Transactions on Mathematical Software*, Vol 5(1), pp. 1–17, 1979.

[99] Zienkiewicz O.C. and Taylor R.L. *The Finite Element Method. Volume 1 The Basis*. Butterworth-Heinemann, 2000.

[100] Zienkiewicz O.C. and Zhu J.Z. 'A simple error estimator and adaptive procedure for practical engineering analysis.' *International Journal for Numerical Methods in Engineering*, Vol 24, pp. 337–357, 1987.

[101] Zienkiewicz O.C. and Zhu J.Z. 'The superconvergent patch recovery and a posteriori error estimates. part 1: The recovery technique.' *International Journal for Numerical Methods in Engineering*, Vol 33, pp. 1331–1364, 1992.

[102] Zienkiewicz O.C. and Zhu J.Z. 'The superconvergent patch recovery and a posteriori error estimates. part 2: Error estimates and adaptivity.' *International Journal for Numerical Methods in Engineering*, Vol 33, pp. 1365–1382, 1992.