NUMERICAL TECHNIQUES FOR THE INITIAL VALUE PROBLEM FOR TWO BLACK HOLES

by

SCOTT ALAN KLASKY, B.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 1994

To my parents Charles and Mary Klasky, and my fiancé Jennifer
for all their love, encouragement and support.

Acknowledgements

I wish to thank Matt Choptuik for all the help and support that he has given to me. I think that Matt helped and encouraged me more than anyone else that I have met. This dissertation is more or less a continuation of Matt's Masters thesis. Matt's guidance is in some sense *unprecedented*. Throughout the years of working on the projects for this dissertation, Matt's advice was usually correct, and always there. Matt even spent the better half of his European trip correcting the many mistakes in this dissertation, I would recommend him to any interested student!

I would of course like to thank my advisor, Richard Matzner, who was always there to help me. It was Richard who convinced me to stay a graduate student, when I was about to give up. I would also highly recommend Richard as an advisor.

I would also like to thank many others who have all helped me get through this, although not nearly in the extent of Matt's help. First I would like to thank Jennifer, who has constantly encouraged me to work hard. I would then like to thank my parents who have constantly supported me, both emotionally and financially.

I would then like to thank my brother for constant encouragement through my college years. I also thank Jennifer's parents, Doyle and Sonja, who welcomed me into their family, and made Texas feel like home for me.

I also thank Armondo Colorado for all his "training". Armondo taught me how to concentrate on the problems that I was facing. He is also the one who got me to lift weights that were three times my body weight. I would also like to thank Reid Guenther for being my best friend, and for being a constant factor in ensuring my happiness here at U.T. I also want to thank Peter Anninos who has also been a good role model, and friend for me. Pete helped me many times, and has always been there for me.

Next I would like to thank Mijan Huq would helped me out with many computer problems. Mijan was also a great "guinea pig" for reading this dissertation. Don Salisbury was also great help in proof reading reading sections of this dissertation. Doug Cline was also a great help in gaining my interest in massively parallel computing. I also want to thank Jan Duffy for all the help she has given me through the years to help get through all the paper work associated with being a graduate student. I would also like to thank Robert Harkness for teaching me about the Cray's architecture, and helping me proof read multiple copies of this dissertation. He helped me with many of the projects, especially with vectorizing some of the codes.

I would especially like to thank Joan Centrella for introducing me to the field of numerical relativity. Through her support and guidance, I became a graduate student under Richard.

NUMERICAL TECHNIQUES FOR THE INITIAL VALUE PROBLEM FOR TWO BLACK HOLES

Publication No. ____________________

Scott Alan Klasky, Ph.D.
The University of Texas at Austin, 1994

Supervisor: Richard A. Matzner

I investigate numerical algorithms to solve the initial value problem for two black holes. We study black hole collisions for three main reasons. First, black hole collisions should be efficient generators of gravitational radiation and are prime candidates for detection by gravitational wave observatories. Second, black holes are the "point masses" of general relativity, making studies of their interactions as relevant as the Kepler problem is to Newtonian gravity. Thus, there is a high probability that we may discover new information concerning the gravitational interaction. Third, this is a difficult problem for numerical relativists and an apparently impossible problem without a numerical approach. A major goal of numerical relativists is to compute "general" solutions to the Einstein equations, and the ability to solve the black hole problem will take us much closer to this goal. Analytical solutions to the Einstein equations are unavailable for strong-field and highly time-dependent regimes. These equations are a set of non-linear, coupled, hyperbolic, and elliptic PDE's, which must be solved numerically. I use the space-plus-time approach which separates the problem into an initial value problem and an evolution problem to generate future (or past) data.

In this dissertation I solve one of the initial value (constraint) equations using finite differencing in Cartesian coordinates. This coordinate system allows for multiple black holes and contains no coordinate singularities. One of my major goals was to overcome the difficulties associated with using this coordinate system, since the coordinate surfaces are not coincident with the domain boundaries.

This dissertation is mainly concerned with discovering and eliminating deficiencies of my first code written to solve the Hamiltonian constraint equation. In doing so I have found several empirical schemes to generate extrapolatable solutions, needed to provide extremely

accurate initial data for wave prediction. In particular I develop extensions to Multi Level Adaptive Techniques to generate extrapolatable results.

With these new techniques I will be able to create "general" solutions to the black hole initial value problem to a specifiable accuracy. These techniques will be of substantial importance in evolving black hole collisions to prepare a library of expected gravitational radiation waveforms.

Table of Contents

Appendix

List of Tables

List of Figures

Chapter 1

Introduction

1.1 General overview

Einstein's general theory of relativity is perhaps the most elegant theory in all of physics. Proposed in 1915 by Albert Einstein, it has forever changed our understanding of the nature of gravity and how it shapes our universe. The beauty of this theory is that it links gravity with space and time, in which the curvature of space-time is a direct consequence of gravity. Many predictions from this theory have been experimentally verified to better than one percent [53][54]. The classical tests of general relativity, namely the deflection and time delay of light and the perihelion shift of Mercury, test only the weak-field and stationary features of the theory. Even the binary pulsar PSR 1913+16 is not a test of general relativity in a fully dynamical field regime, but we are able to indirectly observe gravitational waves from it. There are, however, no direct theoretical experimental tests of the dynamical strong-field limits of general relativity. The direct detection of gravitational radiation will provide the means to more fully test Einstein's theory, and will eliminate skepticism of Einstein's theory that persists because we still have not been able to detect gravity waves. From a theoretical point of view, the details of gravitational waveforms can be used as an additional test for general relativity and other alternative theories of gravitation.

For more than 30 years, experimentalists have been working on gravitational antennas, but there has been no detection to date. The latest generation of detectors use Fabret-Perot interferometers, the largest of which will be known together as the Large scale Interferometer Gravity wave Observatory(LIGO). LIGO is designed to be sensitive enough to have a high probability of detecting gravitational waves emitted by the last few minutes of neutron stars or black hole infall. As discussed by Thorne [48], the most obvious candidate for source of gravitational radiation is the collision of two black holes. This type of event should be the most astrophysically "clean" source for gravitational signals. The strongest gravitational wave production should arise from non-head on collisions, a fully three dimensional situation, and a very difficult one to model, requiring advanced numerical techniques. This dissertation deals with some of these techniques which we believe will aid in the quest of modeling general relativity in fully three dimensional situations.

When dealing with numerical computations in physics, we are faced with the dilemma of choosing between accuracy of the solution and available computational resources (*i.e.* CPU time and memory). This is because, in general, when we model the equations on the computer, we discretize the system onto some mesh, via finite differencing, finite elements, or finite volume. The finer meshes generate more accurate results, but use more CPU time and more memory. Since we are limited in CPU time and memory, we can only discretize the system to some degree which will limit the accuracy of the solution. Complex problems which model three spatial dimensions use tremendous amounts of computer memory and time, and therefore must use efficient algorithms and powerful supercomputers. The goal of this dissertation is to develop algorithms which are optimally efficient, where the amount of computational work is proportional to the amount of real physical change in the computed systems. This is what Achi Brandt terms his "golden" rule[9] .

In order to deal with very large high performance computers, we must deal with the issue of parallelization. Today's supercomputers are no longer single CPU's, but rather massively parallel systems with numerous CPU's. [1] This issue of parallelization further complicates the development of computer codes, since we must worry about scalability; *i.e.* we want a code to run twice as fast if we use twice the number of processors.

1.2 Evolving black holes

The pursuit of numerical solutions to Einstein's equations first began in the 1960's when spherical collapse problems were carried out by May and White (1966),and others. Hahn and Lindquist [30] (1964) attempted the first computation of the coalescence of two black holes, but because of the lack of sufficient computer resources, their attempt was only partially successful. It was not until the mid 1970's that Smarr, Čadež and Eppley revisited the 2-black hole problem. Smarr explored the head-on collision of two black holes. During this time, sufficient memory and computational speed was available to work with axisymmetric black hole encounters; head-on non-spinning black holes. This was possible because this problem has only 2-spatial dimensions, (evolving in time) which requires far less computer time and memory to model than cases which vary in all three spatial dimensions. We shall consider time dependent codes. Thus, our 3-dimensional code is often referred to in the literature as a (3+1)-dimensional code, the (+1) denoting time dependence.

Smarr and Eppley [46][25] (1975) succeeded in evolving the collision and coalescence of two black holes and obtained estimates of the gravitational waveforms emitted. Today we are looking to evolve two black holes with non-aligned momenta and non-aligned spins, a

[1] As of today, the largest MIMD machine is over 1800 CPU's on a Intel Paragon at Sandia Laboratory

generically 3-dimensional problem. We need these computations in order to prepare a library of waveforms expected for LIGO.

1.3 Choice of coordinate system

The vast majority of numerical research on the collision of two black holes performed to date has used the Čadež coordinate system [19]. These coordinates are shown in Figure 1.1 [20]. The Čadez coordinates are body-fitted coordinates that become spherical at large distances from the black holes and, "split" near the holes to encircle each throat. Thus in Figure 1.1, we see two half circles where the two throats are, and between the holes we see a singular point. Although Čadez coordinates are ideal for numerical studies of two black holes, they cannot be extended to configurations containing more than two black holes. These coordinate grids contain coordinate singularities which can cause problems for numerical evolution schemes. Using Cartesian coordinates eliminates both of these problems since there are no coordinate singularities and any number of black holes can, conceivably, be handled. One of the key features of the Cartesian approach is that points of the lattice (3.1) are used to approximate the inner boundaries (i.e. the "holes") of the solution domain. Figure 1.2 illustrates a typical computational domain which might be used for the case of a computation involving two holes in two dimensions. The throats of the two holes are represented by the large circles, and the points in the domain are labeled by the smaller circles. The small dots are those points which are not in the computational domain. Here we see the biggest disadvantage of rectangular coordinates, namely there are very few points exactly on the boundaries. So, the coordinate surfaces will no longer be coincident with the domain boundaries at the black holes, making accurate imposition of boundary conditions much more difficult than with adapted coordinates, such as Čadež coordinates. Cartesian coordinates also tend to cover the coordinate surfaces less efficiently than do Čadež coordinates. One way around this issue is to use compactified coordinates, since this will then allow the grid spacing around the holes to be very small, whereas far away from the holes the grid spacing would be very large. Here, a compactified Cartesian coordinate system will allow us to set the outer boundary to large distances from the strong field regime while maintaining high accuracy near the black holes [15]. The problem with compactified coordinates is that the effective physical spacing increases towards the outer boundary, and the information concerning gravitational waveforms will be lost. Thus, there is apparently a limit on how coarse the zoning can be. We believe that by using a rectangular coordinate system with no coordinate singularities along with adaptive gridding techniques, we should be able to develop highly accurate evolution schemes to evolve the extremely accurate initial data we can already generate[20]. The adaptive gridding techniques will dictate where more grid points will lie. Thus, more grid points will lie close to the throats, and fewer points will be used far away from the throats.

1.4 Survey of existing work of the initial value problem

Einstein's equations are tensor equations which treat the entire four dimensional spacetime concurrently. Arnowitt, Deser and Misner (ADM)[1] decomposed this spacetime into a set of space-like slices stacked together and labeled by a time coordinate. We refer to this decomposition as the 3+1 ADM formalism. Using this formalism it has been shown that general relativity has a well posed initial value problem, *i.e.* one can find initial value data on a space-like slice that can be evolved using the geometrodynamic equations of motion of the 3+1 ADM formalism, to generate the complete spacetime. Unlike other theories in physics which allow us to freely specify the initial conditions of a situation, general relativity dictates constraints in which not all of the degrees of freedom are freely specifiable. Work on the initial value problem began in 1944 by A. Lichnerowicz[35]. Years later Choquet[17], York and his collaborators[57][5][7][6][33] decomposed spacetime into a foliation of spacelike slices, and formulated the initial value problem. The key feature of York's procedure is that it provides us with a well-defined prescription for which some degrees of freedom are to be specified, and other degrees of freedom must be determined from the solution of the constraint equations. These constraint equations are the initial value problem.

With York's formalism, one has a "simple way" of calculating the initial data with non-zero linear and angular momentum. Once York formulated his scheme for solving "the initial value equations", many investigators [47][19][12][56] computed numerical solutions describing data for two black holes. For example, York [57], Bowen[4], and Bowen and York [7] developed a formalism for specifying part of the initial data for a single black hole with non-zero linear and angular momenta. Later, many authors, York and Piran[56], Choptuik[15], Rauber[43], Thornburg[47], Cook and York[21], studied single-hole, axisymmetric initial-data sets based on this framework which used numerical methods to solve the Hamiltonian constraint. In 1983, Kulkarni *et al.*[33] developed a formalism for prescribing the initial data for multiple black holes. Based on this formal approach, Bowen, Rauber, and York [55] detailed an approach for specifying the initial data for two black holes with axisymmetric parallel spins. Rauber[43] attempted to solve these equations numerically, but was unable to generate any complete solution.

In was not until 1993 that three independent groups generated initial data for two black holes with general spin and linear momentum [20]. It was clear that all three of the methods were able to construct useful initial data sets to be used in the simulation of black-hole collisions. The most accurate solutions were generated by Cook's[20] code which solved the finite differenced [2] 3+1 Hamiltonian constraint [3] equation on a Čadež coordinate grid;

[2] Finite Difference methods are a way to generate discrete versions of differential equations using a set of Taylor series approximations, truncated to some order, to approximate the differential equation.

[3] The initial value problem involves the solution of the momentum and Hamiltonian constraint

the difference equations were then solved via the multigrid algorithm[8]. Choptuik, Klasky, and Matzner also used finite difference techniques, but this time on a uniform Cartesian grid. The resulting equations were solved using a variant of line successive overrelaxation (LSOR)[58]. The approach developed by Dubal, Oliveria and Matzner[20], used a global, spectral-like method known as the multiquadric approximation scheme. Here, functions are approximated by a finite sum of weighted quadric basis functions which are continuously differentiable. An important result of this paper was that Cook's code was precisely second order. Second order solutions have errors which have a leading error term which is in the square of the discretization scale. These errors usually result from truncating Taylor series expansions to generate the difference equations. This allowed a kind of extrapolation, Richardson extrapolation to achieve errors which were demonstrably fourth order in the discretization scale, h. Another important result of this paper was that the solutions obtained using Cartesian coordinates were successful in modeling the "non-trivial" boundary topologies. The multi quadric method was an intriguing alternative approach, but there were two main drawbacks: the ill-conditioning of the matrix system to be solved, and the lack of formal numerical analysis.[4]

1.5 The goals of this dissertation

In this dissertation we describe numerical techniques which we will use to construct three-dimensional initial data for the collision of two black holes. In particular we look at the solution of the Hamiltonian constraint equation which is an elliptic partial differential equation with Robin boundary conditions. Here we also examine general linear elliptic systems with a large number of unknowns. Our goal is to develop techniques to solve elliptic systems using Multi Level Adaptive Techniques and obtain solutions which can be extrapolated to fourth order accuracy. In order to achieve this goal, we find it useful to look at systems which are more simple than the 3d Hamiltonian constraint equation.

We use Cartesian coordinates although there are clearly two major disadvantages using these coordinates instead of Čadež coordinates. The first disadvantage is that the coordinates do *not* conform to the inner boundaries ("holes") of the problem domain. This means that the formulation of accurate differenced versions of the boundary conditions is considerably more involved here than those formulations which use Čadež coordinates[20] . Secondly, because we use a *uniform* Cartesian grid (*constant* mesh increment, h, in each of

equations[57]. There is a prescription[19] which determines the momentum constraint for two black holes, but the Hamiltonian constraint equation involves the solution of a partial differential equation which we carry out numerically.

[4] What we mean by the lack of numerical analysis for this approach, is that when one increases the number points used in calculation, there is no formalism developed to tell one how fast the error will be reduced.

Figure 1.1: A Čadež coordinate system.

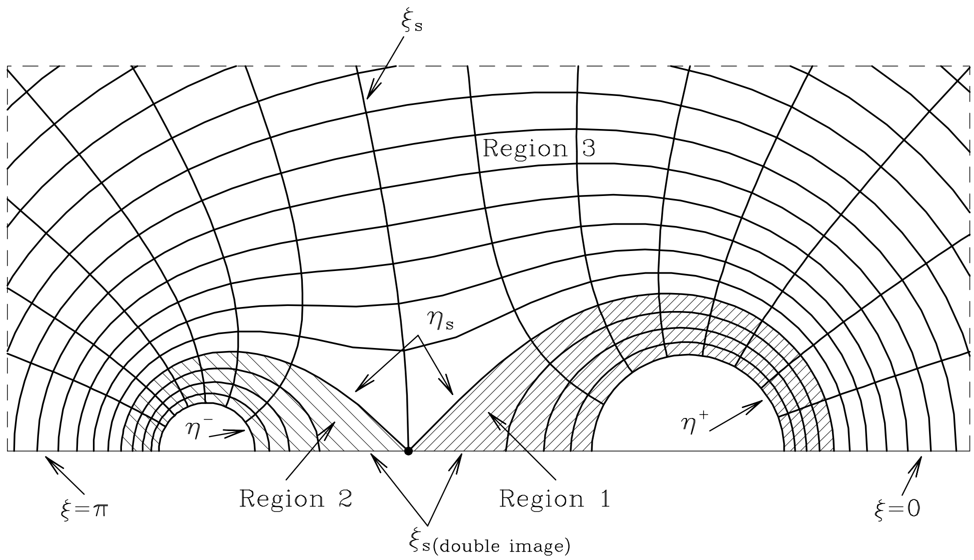
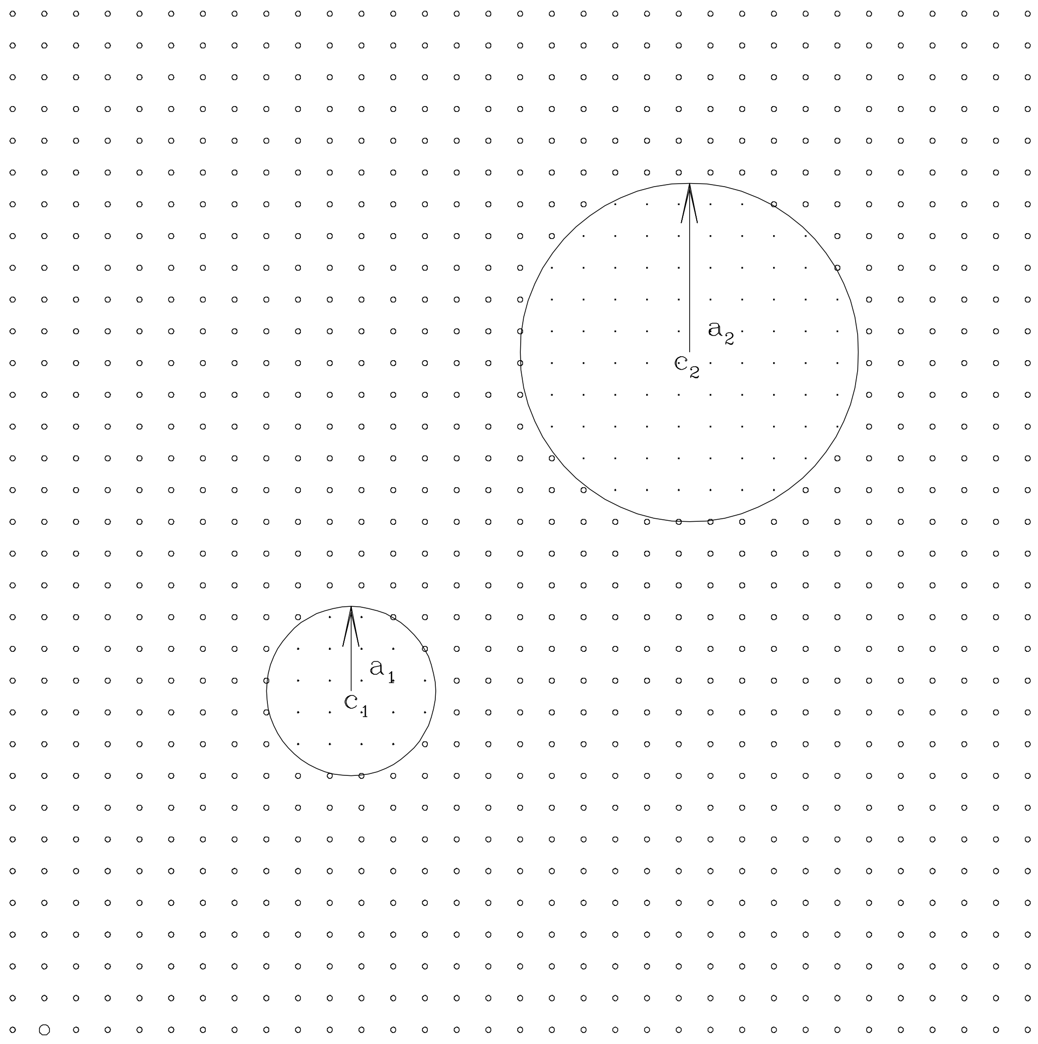


Figure 1.2: A Cartesian coordinate system.



the coordinate directions), the combination of 1) the need to resolve steep gradients near the holes and 2) limitations on our computational resources (memory and time), places a severe restriction on the radius at which the outer edge of the computational domain is located. In practice this means that we must use an adaptive mesh in order to impose the asymptotic boundary condition (2.34) far away from the holes.

Our hope is that by working with the initial value equations we can more readily address the complications induced by the non-trivial boundaries than we could by looking at them in an evolution code. The goal is not to produce the most accurate code to solve the initial value equation, since Cook already has a code that will solve to extrapolatable second order accuracy, but rather to develop numerical techniques which we should be able to use in a Cartesian evolution code. Another goal is to incorporate the most powerful numerical solver for elliptic equations, multigrid, to solve our equations. All of the codes that we use are coded in `FORTRAN 77`.

In Chapter 2, we give a brief discussion of the initial value problem for general relativity. We start with the ADM formalism, then use York's formalism to rewrite the initial value problem as a coupled quasi-linear set of elliptic partial differential equations (PDE's) for four "potentials" which can be viewed as the general relativistic generalization of the familiar Newtonian gravitational potential. Next, the two constraint equations which must be solved in order to obtain initial data for the two black holes are presented[22] .

In Chapter 3, we discuss the basic numerical analysis needed to solve elliptic partial differential equations. It is in this section that we will attempt to understand the need to use the most powerful numerical techniques to solve these equations. These more powerful techniques must be used in order to obey Brandt's golden rule[9] and because the coordinate surfaces are not coincident with the domain boundaries at the black holes. We introduce basic concepts of finite differencing and present some traditional methods of solving elliptic equations[58] [50]. In particular, we present the "art" of finite differencing around boundaries, which is much more difficult than differencing around interior points. We present only the traditional iterative methods for solving elliptic equations, and make no attempt to detail any of these techniques. These traditional methods are conceptually easier to understand than the more modern techniques, but unfortunately they do not obey Brandt's golden rule. In particular, we use one of these methods (Line SOR) to solve the Hamiltonian constraint for two black holes [20].

In Chapter 4, we discuss the multigrid method for solving elliptic equations. The multigrid method is an algorithm which uses a series of meshes to solve an elliptic partial differential equation. We first describe the basic philosophy of multigrid, and then discuss how one can develop and apply this technique in a general problem. Next we introduce

Richardson extrapolation, which is a technique that can be used to obtain fourth order accurate solutions from second order accurate solutions. Deferred correction is also introduced. This is a technique which allows one get more accurate, higher order, solutions, from low order solutions. Finally we will describe the Adaptive Multiple Refinement (AMR) scheme based on the Berger and Oliger[2], and Choptuik[15] algorithm.

In Chapter 5 we present our three dimensional, two black hole code. Here we examine some solutions obtained using this code and compare our results to results from other published codes. We also present convergence studies to show that this code is only first order accurate.

In Chapter 6 we present a series of test problems that address different questions concerning the development of an adaptive multigrid code in three dimensions for two black holes. The first test is a model problem that focuses on the multigrid methodology only. The second problem addresses the development of adaptive codes. Here we examine issues such as the cost of computation and data structures. A third test examines the asymptotic condition, equation (2.34), which requires that the spacetime be flat far away from the black holes. This test not only allows us to see the difficulty in implementing non-trivial boundary conditions in a multigrid algorithm, but also it implements deferred correction and attempts to obtain extrapolatable solutions from this algorithm. The fourth problem examines the issue of Richardson extrapolation using a adaptive multigrid code, explained in Chapter 4. The fifth problem looks at the problem of designing an efficient multigrid code in one dimension, when the grids are non-uniform. The sixth problem combines the previous three to design a general one dimensional multigrid code which has Robin boundary conditions. [5] Finally, the last problem examines the development of a two dimensional multigrid code for a single black hole.

Chapter 7 summarizes our work and examines some of the critical issues that need to be further investigated in future work.

Finally, we present a parallel implementation of a 1D multigrid algorithm. Since computers of today are using multiple CPU's, we feel that we should examine the issue of parallelization of multigrid algorithms. In particular, we examine the overall effectiveness of our implementation.

[5] Robin boundary conditions are differential equations which include the function and its first derivative.

Chapter 2

The initial value problem for 2 black holes

This chapter provides a brief introduction to the initial value specification formalism. This is not complete by any means, and we refer the reader to the vast amount of literature written on this subject[22] [15] [57] [5] [26].

2.1 Introduction

One of the most outstanding achievements of the general theory of relativity is that there is no longer some preferred notion of time. In this chapter we will demonstrate how to decompose spacetime into space plus time in a general way using the Arnowitt-Deser-Misner (ADM)formalism[1], allowing space to be separated from time. The need to do this becomes apparent when we attempt to solve Einstein's equations numerically using the space + time technique. If we use the various "null cone approaches" then we do not need to split space and time. This procedure is well established and we will give a brief summary of only the major steps necessary to derive the initial value equations for two black holes[31]. We show how the ADM split of spacetime results in a set of Cauchy surfaces parameterized by time. Thus, the spacetime manifold M is separated into a family of spatial hypersurfaces Σ_t. Next we will follow Wald's approach to derive the constraint equations[53] . We then outline York's procedure that, when applied to the constraint equations, results in a set of equations that are of a form particularly amenable to numerical solution. Finally we apply these equations to black hole spacetimes.

2.2 The ADM approach

In order to exploit the special nature of time in Einstein's equations, we break the full, four-dimensional covariance to help reveal their dynamical nature. Here we will use the usual geometric units ($G = c = 1$) and use the MTW[36] sign conventions for the metric and curvature. We also let the tensor indices i, j run from one to three whereas all other indices run from zero to three,(where the zero component is the time-like index). The

3+1 approach to general relativity[2] involves a decomposition of the 4-dimensional spacetime manifold into an infinite family of edgeless 3-dimensional spacelike hypersurfaces; *i.e.* we decompose spacetime into space and time where each hypersurface, Σ_t is at a certain time, t.

This decomposition enables us to break the four-dimensional covariance of Einstein's equations,

$$G_{\mu\nu} = 8\pi T_{\mu\nu}, \tag{2.1}$$

such that it is possible to pose them as a Cauchy problem[22]. Here $G_{\mu\nu}$ is the Einstein tensor, and $T_{\mu\nu}$ is the stress energy tensor. Einstein equations consist of 10 equations, of which 4 are the initial value or constraint equations and 6 are the evolution equations. The initial value equations are

$$\begin{aligned} G_{00} &= 8\pi T_{00} \\ G_{0i} &= 8\pi T_{0i} \end{aligned} \tag{2.2}$$

which provide constraints on the initial data g_{ij}. The reason why these equations are known as constraint equations is that, since these equations involve no second derivatives with respect to time, they represent equations which must be satisfied at all times, *i.e.* constraints. There is nothing surprising about why these equations exist, since there is a direct analogy with Maxwell's equations [56]. The reason why the G_{00} equation represents the Hamiltonian constraint equation is that it involves the energy density. The G_{0i} equation involves the momentum density and thus it is called the momentum constraint equation.

A relationship between the components of the metric of spacetime $g_{\mu\nu}$ and those of the spatial 3-metric γ_{ij} in the 3+1 split must be defined. The standard Arnowitt-Deser-Misner (ADM) form for this metric is,

$$ds^2 = -\alpha^2 dt^2 + \gamma_{ij}\left(dx^i + \beta^i dt)(dx^j + \beta^j dt\right), \tag{2.3}$$

where the functions α and β^i are called the lapse and shift functions, respectively. Here, α determines the lapse in proper time experienced by an observer moving between nearby hypersurfaces in the direction normal to the surfaces while β^i describes the shift in the spatial coordinates between the two slices relative to normal propagation. We see that the introduction of the lapse and shift are simply a manifestation of the fact that we have chosen a particular coordinate system to label the events of our spacetime, (see Figure 2.1). We will now follow Wald's[53] derivation in generating the initial value equations. Since the Σ_t are level surfaces, at each point it is possible to construct a one-form,

$$n_a = -\alpha\nabla_a t, \tag{2.4}$$

where

$$\alpha = -t^{\alpha} n_a = (n^a \nabla_a t)^{-1} \tag{2.5}$$

and ∇_a is the covariant four derivative.

2.3 Projection tensors and the covariant derivative

Now that we have decomposed space-time into space and time, we define the projection tensor needed to project an arbitrary vector into the hypersurface, Σ_t. So, on every Σ_t we define the projection tensors:

$$\begin{aligned} \perp^a_b &\equiv \delta^a_b + n^a n_b, \\ N^a_b &\equiv -n^a n_b. \end{aligned} \tag{2.6}$$

The projection tensor, $\perp^a_b$, projects the free indices of a given tensor into Σ_t. Similarly N^a_b projects the free indices normal to Σ_t. The projection tensor $\perp$ induces a 3-metric from the 4-metric:

$$\gamma_{ab} = \perp^c_a \perp^d_b g_{cd} = (g_{ab} + n_a n_b). \tag{2.7}$$

where γ^{ab} is the inverse of γ_{ab} and $\gamma_{ab|c} = 0$, where $|$ denotes a covariant derivative in Σ_t.

The 3-covariant derivative D induced on Σ_t, by ∇ is obtained by projecting every index of the tensor produced by ∇ onto Σ_t. For instance,

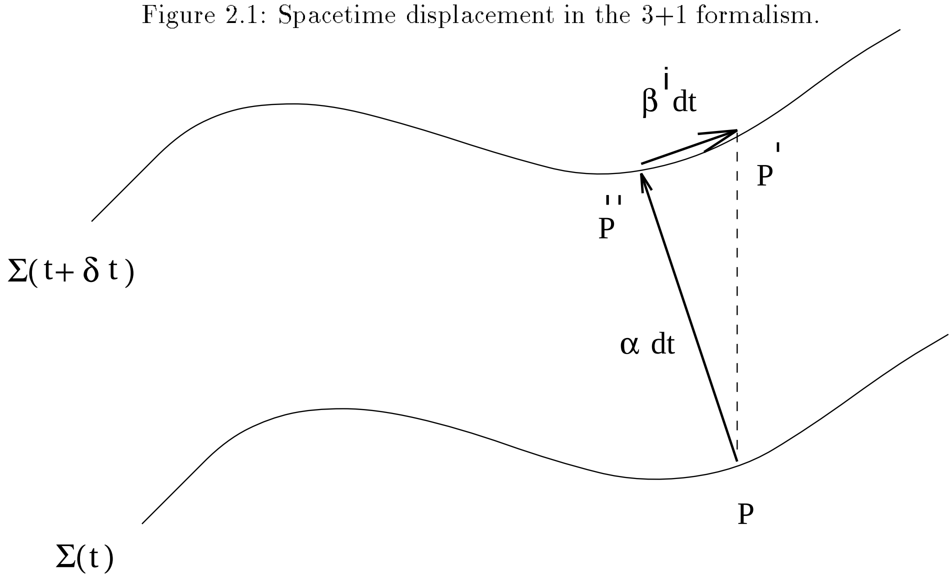
$$\begin{aligned} D_a f &= \perp^b_a \nabla_b f, \\ D_a v^b &= \perp^c_a \perp^b_d \nabla_c v^d, \\ D_a T^b_c &= \perp^d_a \perp^b_e \perp^f_c \nabla_d T^e_f. \end{aligned} \tag{2.8}$$

2.4 The constraint equations

We will start this section by defining the extrinsic curvature, and then proceed by deriving the constraint equations. The extrinsic curvature tensor is a spacelike object which determines the contraction and shear of the normals to Σ_t . Thus, its trace is the average contraction of these normals. The extrinsic curvature can be thought of, roughly, as the "velocity" of the 3-metric. We let v^a and w^b be two vector fields, which we shall project on the hypersurface using $\perp$,

$$\hat{v}^a = \perp^a_b v^b \quad \text{and} \quad \hat{w}^b = \perp^b_c w^c$$

Figure 2.1: Spacetime displacement in the 3+1 formalism.



Then, the extrinsic curvature of Σ_t is defined by

$$\begin{aligned} \mathbf{K}(\hat{\mathbf{v}}, \hat{\mathbf{w}}) &= -(\nabla_a n_b)\, \hat{v}^a \hat{w}^b \\ &= \frac{1}{2} (\mathcal{L}_{\mathbf{n}} \gamma_{ab})\, \mathbf{v}^a \mathbf{w}^b \end{aligned} \tag{2.9}$$

where $\mathcal{L}_{\mathbf{n}}$ is the Lie derivative along $\mathbf{n}$. The Lie derivative , $\mathcal{L}_{\mathbf{n}}$ describes the flow of points in the direction of the vector field $\mathbf{n}$.: . To derive the Hamiltonian constraint equation, we expand $\gamma^{ac}\gamma^{bd}R^{(4)}_{abcd}$ as

$$\begin{aligned} g^{ac}g^{bd} \perp_a^m \perp_b^k \perp_c^l \perp_d^s R^{(4)}_{mkls} &= \gamma^{ml}\gamma^{ks} R^{(4)}_{mkls} \\ &= \left(g^{ml} + n^m n^l\right)\left(g^{ks} + n^k n^s\right) R^{(4)}_{mkls} \\ &= R + 2R_{ab}n^a n^b \\ &= 2G_{ab}n^a n^b. \end{aligned} \tag{2.10}$$

The first Gauss-Codazzi equation[53] is

$$\perp_a^m \perp_b^k \perp_c^l \perp_d^s R^{(4)}_{mkls} = R_{dcba} + K_{ac}K_{bd} - K_{bc}K_{ad} \tag{2.11}$$

which is identical to the left hand side of equation (2.10).

Using the Einstein field equations (2.2) and the first Gauss-Codazzi equation (2.11), we can write the Hamiltonian constraint in the form

$$R + K^2 - K_{ab}K^{ab} = 16\pi\rho, \tag{2.12}$$

where

$$\rho = T_{ab}n^a n^b,$$

is the local energy density.

We can derive the momentum constraint equations by looking at

$$\begin{aligned}\perp_a^m n^n G_{mn} &= \perp_a^m n^n \left(R^{(4)}_{mn} - \frac{1}{2} g_{mn} R \right) \\ &= \perp_a^m n^n R^{(4)}_{mn} \end{aligned} \tag{2.13}$$

and the second Gauss-Codazzi equation

$$N_a^m \perp_b^n \perp_c^o \perp_d^p R^{(4)}_{mnop} = n_a \left(D_d K_{cb} - D_c K_{db} \right). \tag{2.14}$$

where again N_a^m is the projection tensor. If we act on both sides of equation (2.14) with $n^a g^{bc}$ we get

$$n^m \perp_d^p R^{(4)}_{mp} = D_c K_d^c - D_d R_c^c \tag{2.15}$$

which we can equate with equation (2.13) to derive the momentum constraint

$$D_c K_d^c - D_d K_c^c = 8\pi n^n \perp_d^m T_{mn}. \tag{2.16}$$

Finally, we use

$$j^i \equiv \gamma^{ik} j_k \tag{2.17}$$

where

$$j_k \equiv -n_a T_k^a \tag{2.18}$$

to write the momentum constraint in the form

$$D_j \left(K^{ij} - \gamma^{ij} K \right) = 8\pi j^i. \tag{2.19}$$

This equation is a vector equation, which is equivalent to 3 scalar equations. Here j^i is the momentum density.

2.5 Conformal transformation of the constraints

In order to make the constraint equations easier to model numerically, we will use the York[57] procedure of conformal transformations. York[57] provided a constructive way to solve the constraint equations that involves solving elliptic partial differential equations.

A key idea (originally suggested by Lichnerowicz in 1944)[35] of York's approach is to specify the physical data up to conformal equivalence. York's contributions (along with various coworkers)[57] include applying the conformal and transverse-traceless decomposition of the constraint equations, as well as developing a method of imaging applicable to tensors[22] . The chief success of York's work is that it puts the initial value equations into a form which allows them to be solved with numerical methods, and in some cases, by analytic means. Before applying this formalism, we will first choose a coordinate system with a basis $(\mathbf{n}, \mathbf{e}_i)$ where $\mathbf{e}_i$ are spatial vectors, and $\mathbf{n}$ is a time-like vector. Hereafter, we make the following identifications

$$\gamma_{ab} \Longrightarrow \gamma_{ij}$$

$$\perp^a_b \Longrightarrow \delta^i_j$$

$$D_a \Longrightarrow \nabla_i .$$

where we now use latin indices i, j , (which range from 1 to 3) to indicate that we are in the basis.

We write the physical 3-metric in the conformal form

$$\gamma_{ij} = \psi^4 \hat{\gamma}_{ij} \tag{2.20}$$

where $\hat{\gamma}_{ij}$ is a background metric which is to be freely specified. (In the remainder of this chapter, a caret will denote a "bare" 3-tensor which is related to the physical tensor via a conformal transformation. Also, bare tensors have their indices raised and lowered with the bare metric.) All quantities which are functions of the metric have transformation properties which follow from (2.20). In particular we find that

$$\begin{aligned} \Gamma^i_{jk} &= \hat{\Gamma}^i_{jk} + 2\psi^{-1}\left(\delta^i_k \psi_{,k} + \delta^i_j \psi_{,k} - \hat{\gamma}^{il}\hat{\gamma}_{jk}\psi_{,l}\right) \\ R &= \psi^{-4}\hat{R} - 8\psi^{-5}\Delta\psi \end{aligned} \tag{2.21}$$

where $\Delta\psi = \gamma^{ij}\nabla_i\nabla_j\psi$.

The York procedure requires us to specify freely on the hypersurface the value of the mean extrinsic curvature, TrK

$$K \equiv TrK = \gamma_{ij}K^{ij}. \tag{2.22}$$

Since TrK is assumed to be given, it is convenient to isolate the trace-free part, $\hat{A}^{ij}$, of the bare extrinsic curvature, $\hat{K}^{ij}$:

$$\hat{A}^{ij} = \hat{K}^{ij} - \frac{1}{3}\hat{\gamma}^{ij}\hat{K} \tag{2.23}$$

Now Kulkarni[34] showed that for any symmetric rank-2 tensor field $\hat{F}^{ij}$:

$$\nabla_i \hat{F}^{ij} = \psi^{-10}\hat{\nabla}_i\left(\psi\hat{F}^{ij}\right) - 2\psi^{-1}\hat{\gamma}^{ij}\psi_{,i}\hat{\gamma}_{kl}\hat{F}^{kl} \tag{2.24}$$

In particular, $\hat{A}^{ij}$ will have this property. This helps us in deriving a conformal transformation of the constraints since the equations contain terms of this form. Now we make the following transformations:

$$\begin{aligned} A^{ij} &= \psi^{-10}\hat{A}^{ij} \\ A_{ij} &= \psi^{-2}\hat{A}^{ij} \\ j^{i} &= \psi^{-10}\hat{j}^{i} \\ \rho &= \psi^{-8}\hat{\rho}. \end{aligned} \tag{2.25}$$

Using equations (2.25), (2.24) and (2.23), we can rewrite the momentum constraint equation[34] as

$$\hat{D}_j\hat{A}^{ij} - \frac{2}{3}\psi^6\hat{\gamma}^{ij}\hat{D}_jK = 8\pi\hat{j}^i \tag{2.26}$$

and the Hamiltonian constraint as

$$8\hat{\Delta}\psi - \hat{R}\psi - \frac{2}{3}\psi^5K^2 + \hat{A}_{ij}\hat{A}^{ij}\psi^{-7} = -16\pi\hat{\rho}\psi^{-3}. \tag{2.27}$$

Here is where we see the beauty of York's procedure. We are left with four equations which have the freely specifiable quantities: 1) the base 3-metric $\hat{\gamma}_{ij}$, 2) the mean extrinsic curvature TrK, 3) a certain part of $\hat{A}_{ij}$ ([56]) , and 4) the conformally scaled energy and momentum densities $\hat{\rho}$ and $\hat{j}^i$. Once we specify these variables, we can solve the constraint equations for $\hat{A}_{ij}$ and ψ. We can construct the physical initial data using these variables.

2.6 Application of the constraint equations to black hole spacetimes

The initial value equations are simplified when one considers conformally flat, maximal volume, vacuum slices. Conformal flatness means that the physical metric, γ_{ij}, of the hypersurface may be written as

$$\gamma_{ij} = \psi^4 f_{ij} \tag{2.28}$$

where f_{ij} is the flat background 3-metric (previously $\hat{\gamma}_{ij}$) Maximal volume means that

$$TrK = 0.$$

The standard $t=$ constant hypersurfaces in Minkowskii spacetime also have this property.

The vacuum requirement means that there are no matter sources on the hypersurface:

$$\hat{\rho} = \hat{j}^i = 0,$$

and finally we note that conformal 3-flatness implies $\hat{R} = 0$. Thus, the constraint equations take the much simplified form

$$8\hat{\Delta}\psi = -\psi^{-7}\hat{A}_{ij}\hat{A}^{ij} \tag{2.29}$$

and

$$\hat{D}_j\hat{A}^{ij} = 0. \tag{2.30}$$

For the black hole spacetimes that we consider, the topology of the initial-data hypersurface is fixed to be two asymptotically flat universes, *i.e.* they tend to flat spacetimes at large radii, connected by an Einstein-Rosen bridge [24] for each black hole. Each bridge is a connection to another identical universe and we require that the bridges link only two identical universes. To achieve the condition that bridges link only identical universes, an isometry condition[54] must be imposed. This is derived from an inversion symmetry requirement ,which is

$$\psi(r) = \frac{a}{r}\psi\left(\frac{a^2}{r}\right)$$

and has the effect of ensuring an inversion-symmetric solution with $r = a$ as a minimal surface. The solution of the momentum constraint under these conditions was found by Kulkarni *et al.* [34]. Their solution was in the form of an infinite series generated by a method of images applied to tensors. For the case of two black holes, an accurate numerical scheme for evaluating the formal infinite series was developed by Cook[19]. To solve the momentum constraint, we give values for the radius a_α and coordinate center C^i_α, the linear momentum P^i_α and the spin S^i_α, where α labels the hole. Figure 1.2 shows a_α and C^i_α for two holes.

We use a routine developed by Matzner[37] based on Cook's work which determines the solution of the momentum constraint.

Given a solution of the momentum constraints, we then must solve the Hamiltonian constraint with some boundary conditions. The isometry condition applied to the physical metric at a_α yields the following inner boundary condition for the conformal factor

$$n^i_\alpha\hat{D}_i\psi_{|a_\alpha} + \frac{\psi}{2r_\alpha}_{|a_\alpha} = 0 \tag{2.31}$$

where

$$n^i_\alpha = \frac{x^i - C^i_\alpha}{r_\alpha} \tag{2.32}$$

is the unit normal to the surface of the αth isometry surface and

$$r_\alpha = |x^i - C^i_\alpha|. \tag{2.33}$$

The asymptotically flat condition sets the outer boundary condition to

$$\psi \rightarrow 1 \qquad \text{as} \qquad r \rightarrow \infty \tag{2.34}$$

where r is measured from the "hole" singularity, which is somewhat arbitrary in the limit $r \rightarrow \infty$. In practice, it is not often possible to model equations on grids with boundaries set at $r \rightarrow \infty$. However, a multipole expansion can be performed for the conformal factor and truncated at monopole order to generate an approximate asymptotic boundary condition[56]

$$r\frac{\partial \psi}{\partial r} = 1 - \psi + O\left(r^{-2}\right) \tag{2.35}$$

We note that higher order approximate boundary conditions can be derived [19]. We will not use these, however, since our ultimate approach will be to use a very large outer radius where $\psi \approx 1$.

To summarize, our goal is to solve the following equations given values for the radii a_α and coordinate centers C^i_α, the linear momenta P^i_α and spins S^i_α

$$8\nabla^2\psi = -\psi^{-7}\hat{A}_{ij}\hat{A}^{ij} \tag{2.36}$$

and

$$\begin{aligned} n^i_\alpha \hat{D}_i \psi_{|a_\alpha} + \frac{\psi}{2r_{\alpha\,|a_\alpha}} &= 0 \\ r\frac{\partial \psi}{\partial r} &= 1 - \psi \end{aligned}$$

Chapter 3

Numerical techniques

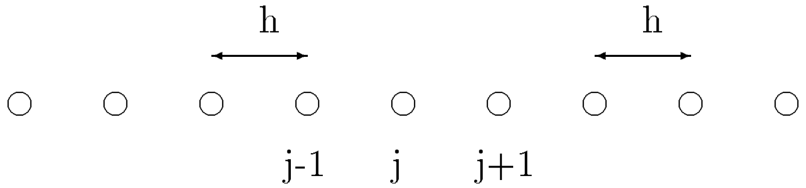
In this chapter, we wish to present our philosophy of numerical analysis. In the first section we present the "art" of finite differencing. We call this an art since there are many ways to finite-difference an equation. Some iterative algorithms will diverge for certain differencing techniques whereas if we use other methods of differencing we could make the algorithms converge.

We use finite differencing techniques, (versus finite element and finite volume methods), since they are often much easier to program than these two other methods. Since the code that will be written to evolve the evolution equations will probably be discretized by finite differencing, we can also gain experience with the differencing around the boundaries of the holes.

After discussing finite differencing, we then present methods of numerically solving elliptic partial differential equations. Here, we look at three "classic" iterative methods,(Jacobi, Gauss-Seidel, and Successive Over Relaxation), although no attempt is made to describe all of the available techniques, nor is any particular method presented in detail. All of these techniques can be written in terms of a general matrix equation. This general matrix equation, is a set of coupled equations, which describe how the solution will change from one iteration to the next. For each method, the matrix eigenvalues will determine the rate of convergence. Later we present interpolation and extrapolation techniques for high order accuracy. We present a general algorithm which will interpolate and extrapolate to fourth order. These interpolation techniques work with all the systems considered in this dissertation. That is, these techniques will be for lattices with holes(spheres), cut out. Next we present further enhancements to the interpolation algorithm, along with some alternative approaches. Finally, we pose a few model problems that we will examine with more elaborate algorithms in Chapter 6. We assume that the reader is already familiar with the basic techniques. [1]

[1] Some classical references to this subject are Young[58], Varga[50], Numerical Recipes[41], and Mitchell and Griffiths[40].

Figure 3.1: A discretized domain with uniform differencing.



3.1 Spatial finite differencing

3.1.1 Basic techniques

In describing spatial finite differencing it is perhaps best to start with a model problem, and then expand from there to handle general problems. Since we do not deal with systems which evolve in time, we will only describe finite differencing for spatial varying systems. In all of our examples we will work in Cartesian coordinates, and use uniform differencing, *i.e.*, constant grid spacing as shown in Figure 3.1.

We will look at two example systems: a one dimensional elliptic PDE, (actually an ODE, but we ignore this for our example), and a two dimensional elliptic PDE. It then should be intuitively obvious how to extend the ideas in this section to three dimensions. The computational domain we use is that of a lattice which has n grid points on an edge. For example, the one dimensional lattice which we use, is defined by

$$\{(ih)\}, \qquad i = 1, 2, \cdots, n \tag{3.1}$$

where h, is the (unique) discretization scale. As a matter of convenience, we will refer to *any* point with coordinates of the form (ih), with i as an integers, as a lattice point, even though it may not be contained in the actual computational domain. We now describe the techniques used to model the following equation

$$\begin{aligned} \frac{\partial^2 u(x)}{\partial x^2} &= f(x) \qquad x \in [0,1] \\ u(0) &= u_0 \\ u(1) &= u_1. \end{aligned} \tag{3.2}$$

This problem can also be represented as

$$Lu = f \tag{3.3}$$

where L represents the differential operator in the system,

$$L = \frac{\partial^2 u(x)}{\partial x^2}, \tag{3.4}$$

Table 3.1: Notation used in this dissertation

$u(x)$	Continuum function
u^h	Discretized function, with grid spacing h
$u^{(n)}$	Discretized function, which is being used in an iterative procedure at iteration number n
$u^h[j]$	One dimensional discretized function with grid spacing h, representing the point at $x = x_j$
$u[j]$	One dimensional discretized function representing the point at $x = j$
$u[J]$	Capital letters generally mean that we are working with coarse and fine grids. The coarse grid functions are denoted with capital letters, and the fine grid have lower case letters.
$u^h[j,k]$	Two dimensional discretized function with grid spacing h, representing the point at $x = x_j, y = y_k$
$u^h[j,k,l]$	Three dimensional discretized function with grid spacing h, representing the point at $x = x_j, y = y_k, z = z_l$
L	Continuum differential operator
L^h	Discretized (difference) operator
$\tilde{u}^h[j]$	The newly updated value of $u^h[j]$ used in an iterative procedure (see $u^{(n)}$)
$:=$	Assignment operator
$a(b)$	$a \times 10^{\mathrm{b}}$

in this case. It is important that the functions involved are smooth, *i.e.* infinitely differentiable, since we will not be able to apply finite differencing if they are not smooth. The first step in defining a finite difference problem is discretizing the system, which means replacing the continuous domain by a discrete domain, as shown in Figure 3.2. Before we describe the discretization process any further, we direct the reader's attention to Table 3.1 which defines the notation which will be used throughout this text for discretized functions and operators. We also put all of our algorithms in a `typewriter` font to emphasize our computer instructions. Each algorithm is pseudo-code taken from our `FORTRAN 77` codes. Where possible, we eliminate most of the clutter in the codes.

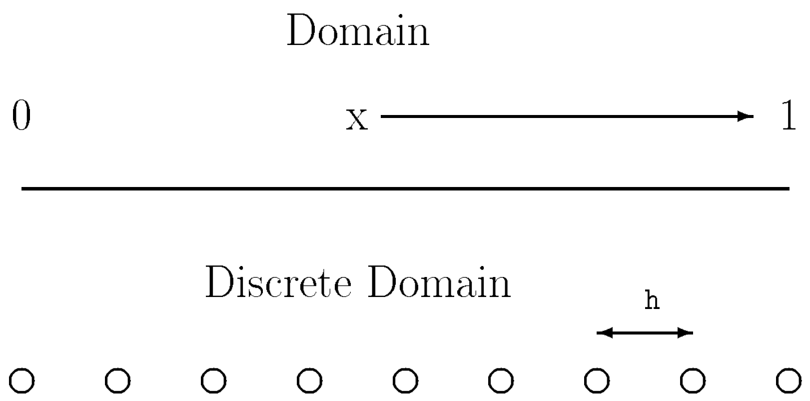
Thus, we discretize the problem domain in the usual manner, by introducing a uniform grid, $x_j = jh$, where $j \in (1, \cdots, N)$ and $h = (N-1)^{-1}$. The finite set of grid points is the discrete domain. Thus, the discretized model problem can now be represented as

$$L^h u^h = f^h \tag{3.5}$$

Given the assumption that u is smooth, Taylor's theorem can be used to approximate the derivatives of u by linear combinations of various grid function values (or nodal values), $u[j]$. For example, for $j = 2 \cdots N-1$ (interior points), we can use the usual second order, centered approximation for second derivatives

$$\frac{\partial^2 u(x)}{\partial x^2}[j] = \frac{u[j+1] + u[j-1] - 2u[j]}{h^2} + O\left(h^2\right) \quad 2 \le j \le N-1. \tag{3.6}$$

Figure 3.2: Replacement of a continuous domain with a finite differenced discrete domain



The model problem uses Dirichlet boundary conditions; thus the one dimensional finite differenced model problem becomes

$$\begin{aligned} \frac{u[j+1]+u[j-1]-2u[j]}{h^2} &= f[j]+O\left(h^2\right)) \quad 2 \leq j \leq N-1) \\ u[1] &= u\left(x\left(1\right)\right) \\ u[N] &= u\left(x\left(N\right)\right) \end{aligned} \tag{3.7}$$

This approximation may be defined from the Taylor series expansions

$$\begin{aligned} u\left(x+h\right) &= u\left(x\right)+h\ u_x\left(x\right)+\frac{h^2}{2}u_{xx}\left(x\right)+\cdots \\ u\left(x-h\right) &= u\left(x\right)-h\ u_x\left(x\right)+\frac{h^2}{2}u_{xx}\left(x\right)+\cdots \end{aligned} \tag{3.8}$$

truncated at $O\left(h^4\right)$. When we use equation (3.7) to represent the second derivative, the leading order error term is in $O(h^2)$. In general we will define the truncation error as

$$\tau^h \equiv L^h u - f. \tag{3.9}$$

The solution error is defined by

$$e^h \equiv u^h - u \tag{3.10}$$

while τ^h is the actual error of the difference solution. We see that the truncation error to the model problem is

$$\tau^h = \frac{h^2}{12} \tag{3.11}$$

which is called second order since the truncation error is of order h^2. We assert (and it can be shown) that the solution of these equations is an approximation to the solution of equation (3.2) whose error is $O(h^2)$ as $h \to 0$. This is perhaps the most commonly used form for second derivatives, but it is important to note that this is not the only second order expression for this term. For example, we can use backward differencing to obtain the formula:

$$u_{xx}[j] = \frac{(2u[j] - 5u[j-1] + 4u[j-2] - u[j-3])}{h^2} + O\left(h^2\right), \tag{3.12}$$

or use forward differencing to obtain:

$$u_{xx}[j] = \frac{(2u[j] - 5u[j+1] + 4u[j+2] - u[j+3])}{h^2} + O\left(h^2\right). \tag{3.13}$$

Higher order approximations, while more accurate for a given grid spacing, in general require more work numerically, (on a per-grid-point basis) both for their evaluation and for the solution of the algebraic equations resulting from their use. If we want the finite differencing to be a higher order, we then must include more points in the stencil. As is often the case in finite-difference work, it is convenient to describe the derivation of our discrete equations in terms of *stencils*. The stencil associated with any given finite-difference expression applied at some generic lattice point (ih) is simply the set of points (or neighborhood) explicitly referenced in that expression. We shall see later that there are often convergence problems when higher order schemes are used with iterative solvers.

Now we shall extend these ideas to a two dimensional model problem,

$$\frac{\partial^2 u\left(x,y\right)}{\partial x^2} + \frac{\partial^2 u\left(x,y\right)}{\partial y^2} = f\left(x,y\right) \quad x \in [0,1], y \in [0,1]. \tag{3.14}$$

Here again, we discretize the system using centered differencing

$$\begin{aligned} u_{xx}[j,k] + u_{yy}[j,k] = h^{-2}\Big(&u[j+1,k] + u[j-1,k] \\ &+u[j,k+1] + u[j,k-1] - 4u[j,k]\Big) \\ &+O\left(h^2\right) \quad j \in [2, nx-1], k \in [2, ny-1]. \end{aligned} \tag{3.15}$$

This is the most commonly used finite difference expression for the Laplacian $\nabla^2 u$, in the special case when the grid spacing in the y direction is equal to the grid spacing in the x direction.

One fast way of determining if a stencil is going to give convergence problems using iterative techniques, is to check for diagonal dominance.[2] We define diagonal dominance for a matrix L with entries $L[j,k]$ by

$$||L[j,j]|| \geq \sum_{k=1, j\neq k}^{n} ||L[j,k]||, \tag{3.16}$$

where at least one row must be have $||L[j,j]|| > \sum_{k=1,j\neq k}^{n} ||L[j,k]||$.

For example in equation (3.15), the off diagonal terms add up to 4, as does the diagonal term. Since the sum of the off diagonal terms is not greater than the diagonal term, we should not expect convergence problems from this equation when we solve for $u[i,j]$. This is known as diagonal dominance. When we look at equation (3.12) we see that the off diagonal terms add up to 10 whereas the diagonal term is only 2. We can expect some convergence problems using most iterative numerical solvers for this equation. Note that we do not claim that diagonal dominance implies fast convergence. Rather, the rule-of-thumb is that diagonally dominant systems can generally be solved iteratively using techniques such as the the Gauss-Seidel method described below. Conversely, when we attempted to solve equation (3.14) using backwards differencing with a Gauss-Seidel algorithm, the iteration did not converge. Thus, when we finite-difference elliptic systems, we will always try to maintain diagonal dominance.

3.1.2 Boundaries

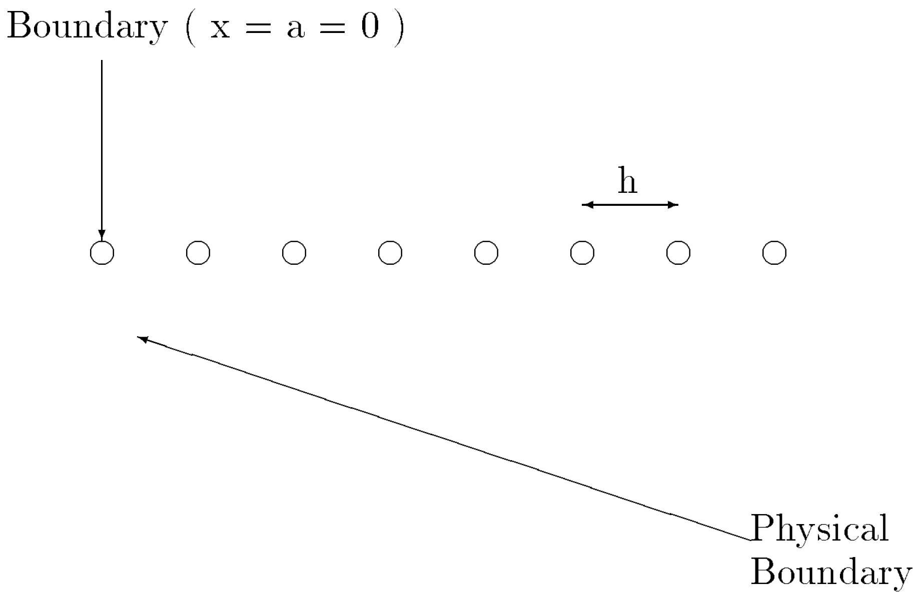
Perhaps the hardest part of finite differencing is working with boundaries. We saw that in the previous section, it was quite elementary to apply finite differencing to generate interior equations. We can always apply second order central differencing for interior equations, which is diagonally dominant. When it comes to treating boundary conditions (or even interior equations in the vicinity of boundaries) we must be very careful in choosing the appropriate finite difference stencil. In this section, we shall first look at a one dimensional example using a general boundary condition where the physical boundary coincides with a lattice point. Next we look at the case when the physical boundary is no longer coincident with a lattice point. After this, we shall look at some two dimensional cases and comment on extensions to three dimensions.

Before we start, we recall that we can easily generate $O\left(h^2\right)$ accurate expressions for use at interior points. Thus, we shall want to be able to use at least $O\left(h^2\right)$ approximations of the boundary equations; in fact our goal will actually be to achieve $O\left(h^4\right)$ accuracy on the boundaries.

---

[2]By convergence problems we mean to say that by not having diagonal dominance, most iterative methods will probably diverge.

Figure 3.3: The 1D discretized domain where $a = 0$.



3.1.3 One dimensional boundary finite differencing

There are three major types of boundary conditions encountered in boundary-value problems: 1) Dirichlet conditions, where the value of the unknown function is given on the boundary, 2) Neumann conditions, where the normal derivative of the function on the boundary is specified, and 3) Robin or mixed conditions, where some combination of the function and its normal derivative on the boundary are given.

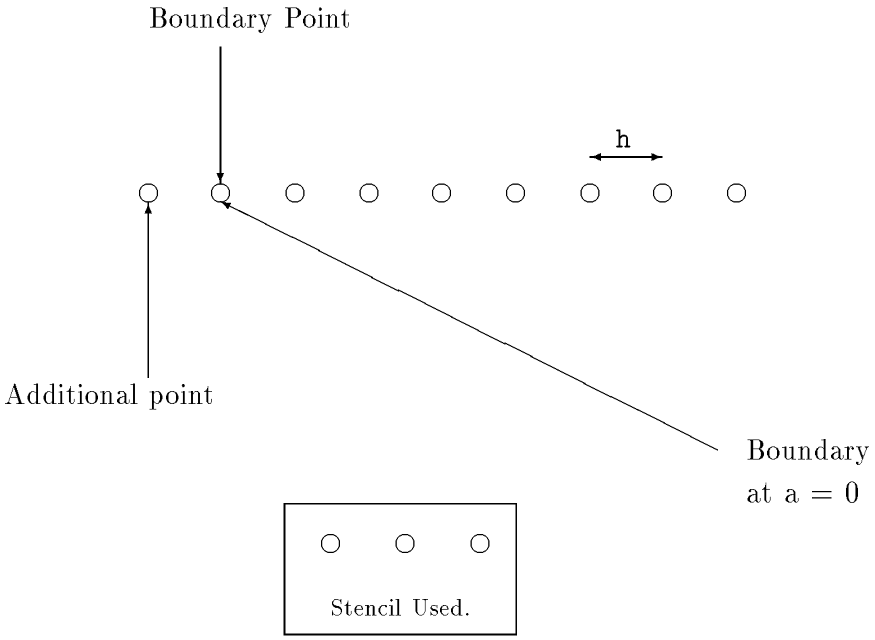
Our general one dimensional boundary condition will be the Robin condition

$$\alpha u + \beta u_x = g \tag{3.17}$$

applied at $x = a$. At first we shall take $x = a = 0$, to explain some simple concepts of finite differencing around boundaries. Next, we apply the boundary condition at some irrational x, so that no lattice point coincides with the boundary. We do this in preparation for extensions into multiple dimensions. Figure 3.3 shows the discretized system for the $a = 0$ case.

There are two main ways of deriving approximations to the boundary conditions. The first method applies discrete forms of both the interior equation and the boundary equation on the boundary by introducing an extra grid point which lies outside the actual continuous domain. The second method keeps the boundary equations separate from the interior equation. Thus, the only difference between these two methods is in the grid structure used.

Figure 3.4: The discretized domain, with the addition of an extra point.



Applying the first method to equation (3.17), we see that we can use a centered finite difference scheme for the boundary equation to get

$$\alpha u\,[j] + \frac{\beta\,(u\,[j+1] - u\,[j-1])}{2h} = g\,[j] + O\left(h^2\right) \tag{3.18}$$

We see that this equation couples the terms at $j-1$ and at $j+1$, although the point at $j-1$ does not exist in the domain. An extra point at this location is then introduced. By introducing another point, we are introducing another unknown, for which we need another equation. There is no problem with this since the usual method for solving this problem is to apply the interior equation (3.6) on the boundary. Using these two equations, we can solve for the boundary value, $u[j=1]$ as follows

$$\begin{aligned}
\frac{(u[j+1] + u[j-1] - 2u[j])}{h^2} &= f[j] + O\left(h^2\right) \\
u[j-1] = h^2\left(f[j]\right) + 2u[j] - u[j+1] &+ O\left(h^2\right) \\
\beta\frac{2u[j+1] - 2u[j] - h^2 f[j]}{2h} + \alpha u[j] &= g[j] + O\left(h^2\right) \\
u[j] = \frac{h}{-\beta + \alpha h}\left(\frac{h}{2}\beta f[j] + g[j] - \beta h^{-1} u[j+1]\right) &+ O\left(h^2\right) \qquad (3.19)
\end{aligned}$$

Figure 3.4 shows a picture of this situation.

The equation for $u[j]$ is also not always diagonally dominant. In this case we obtain the condition for diagonal dominance

$$h \geq \frac{2|\beta|}{|\alpha|}$$

Since h is usually very small, this condition will not be met in general. We also see that we will not be able to apply centered differencing all the time in multiple dimensions, since there are occasions where we must introduce two or more points, and we only have one extra equation. We might think that we can generate extra equations by saying that the function is smooth across the boundary, *i.e*,

$$u[j] = \frac{1}{2}\left(u[j-1] + u[j+1]\right) \tag{3.20}$$

But we will now try to evaluate a first derivative using this averaging condition at the location of u_2 shown in Figure 3.5. We need to consider this case when we look at problems such as the one in Section 6.4, where we are using Adaptive Mesh Refinement (AMR) techniques. The first derivative is evaluated using

$$\begin{aligned}
u\left(x+h\right) &= u + hu_x + \frac{h^2}{2}u_{xx} + \frac{h^3}{6}u_{xxx} + O\left(h^4\right) \\
u\left(x-h\right) &= u - hu_x + \frac{h^2}{2}u_{xx} - \frac{h^3}{6}u_{xxx} + O\left(h^4\right) \\
\frac{u\left(x+h\right) - u\left(x-h\right)}{2h} &= u_x + \frac{h^2}{6}e_3 + O\left(h^4\right) \\
\frac{u\left(x+h\right) + u\left(x-h\right)}{2} &= u + \frac{h^2}{2}e_2 + O\left(h^4\right)
\end{aligned} \tag{3.21}$$

thus, we use

$$\begin{aligned}
u_x &= \frac{u_s - u_1}{2h} - \frac{h^2}{3}e_3 \\
u_s &= \frac{u_3 + u_4}{2} - \frac{H^2}{2}e_2
\end{aligned} \tag{3.22}$$

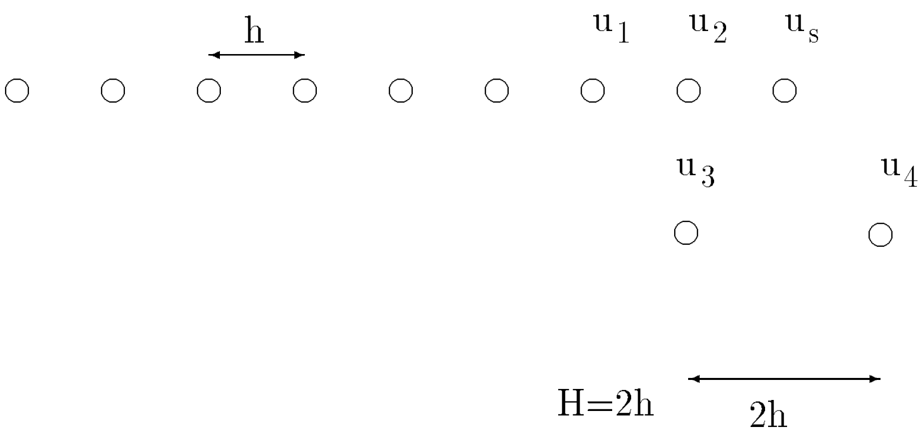
where $H = 2h$,which shows that the equation that would be used for the stencil,

$$\begin{aligned}
u_x &= \frac{\frac{u_3+u_4}{2} - u_1}{2h} - \frac{H}{4}e_2 - \frac{h^2}{6}e_3 \\
u_x &= \frac{\frac{u_3+u_4}{2} - u_1}{2h} + O\left(H\right)
\end{aligned} \tag{3.23}$$

is first order.

The second method is to apply boundary equations to the boundary points, a scheme that we use when we try to solve the system using multigrid techniques, since we treat interior equations separate from boundary equations. This does not mean that we can not use a

Figure 3.5: Trying to get $O\left(h^2\right)$ results from averaging.



multigrid technique to solve the system in the first method, but rather we will only use the second method when we use multigrid techniques. Thus, we use the equation

$$\begin{aligned} \frac{4u[j+1]-u[j+2]-3u[j]}{2h} &= u_x \\ \alpha u[j]+\beta\frac{4u[j+1]-u[j+2]-3u[j]}{2h} &= g[j] \end{aligned} \tag{3.24}$$

for the boundary value. Once again we run into problems. Notice that this stencil will not be diagonally dominant in general, and that we are including more points in the stencil. Through experience, we have learned that trying to come up with diagonally-dominant second order stencils around the boundary, especially in higher dimensions, is a "no win" situation. [3]

In Figure 3.6 the boundary lies between two grid points, a situation we refer to as "straddling" the boundary. In order to derive a second order stencil at this point, notice that we have no use for the interior equation, *i.e.* all we must do is define a set of Taylor expansions around the boundary point. We use

$$\begin{aligned} u_b \equiv u\left(a-h+\xi\right) &= u\left(a\right)+\left(\xi-h\right)u_x+\frac{1}{2}\left(\xi-h\right)^2 u_{xx}+O\left(h^3\right) \\ u_1 \equiv u\left(a+\xi\right) &= u\left(a\right)+\xi u_x+\frac{1}{2}\xi^2 u_{xx}+O\left(h^3\right) \\ u_2 \equiv u\left(a+h+\xi\right) &= u\left(a\right)+\left(h+\xi\right)u_x+\frac{1}{2}\left(h+\xi\right)^2 u_{xx}+O\left(h^3\right) \end{aligned} \tag{3.25}$$

[3] We do not imply that one can not come up with a scheme that will give them second order convergence. What we are saying is that one *could* get into trouble solving these equations.

to obtain

$$u_x = \frac{h-2\xi}{2h^2}u_2 + \frac{2\xi}{h^2}u_1 + \frac{-h-2\xi}{2h^2}u_b \tag{3.26}$$

which is valid to second order, and we use

$$\begin{aligned} u_b \equiv u(a+\xi-h) &= u(a) + (\xi - h)\,u_x \\ u_1 \equiv u(a+\xi) &= u(a) + \xi u_x \end{aligned} \tag{3.27}$$

to obtain

$$u(a) = \frac{\xi}{h}u_b + \frac{h-\xi}{h}u_1. \tag{3.28}$$

Thus, our boundary condition becomes

$$\begin{aligned} g &= \alpha\left(\frac{\xi}{h}u_b + \frac{h-\xi}{h}u_1\right) + \beta\left(\frac{h-2\xi}{2h^2}u_2 + \frac{2\xi}{h^2}u_1 + \frac{-h-2\xi}{2h^2}u_b\right) \\ &= \frac{2\alpha\xi h - \beta(h+2\xi)}{2h^2}u_b + \frac{\alpha(h-\xi)h + 2\xi\beta}{h^2}u_1 + \frac{\beta(h-2\xi)}{2h^2}u_2. \end{aligned} \tag{3.29}$$

Thus we see that there is considerable complexity involved with deriving second order stencils for boundary points of this type, and the extensions into multiple dimensions clearly suggest we abandon these techniques.

Our alternative approach is to discretize the boundaries only up to $O(h)$ and then use deferred correction, discussed in chapter 4 of this text, to get a solution which will be second order accurate on the boundaries. We can easily get a first order version of our boundary equation using forward differencing as

$$\begin{aligned} u_x &= \frac{(u_1 - u_b)}{h} \\ g &= \left(\alpha - \frac{\beta}{h}\right)u_b + \frac{\beta}{h}u_1 \end{aligned} \tag{3.30}$$

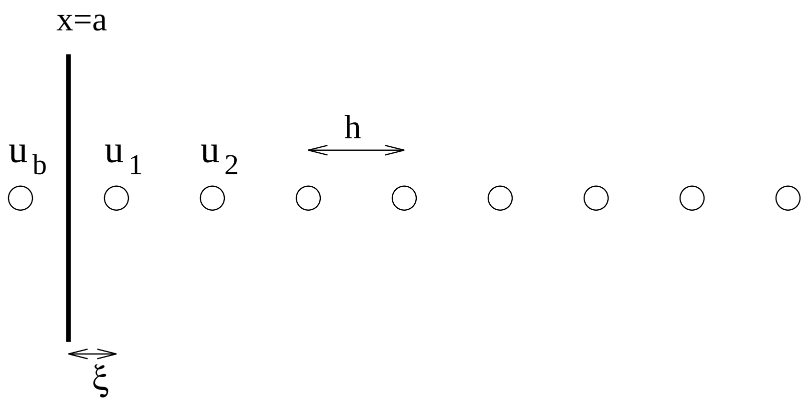
We see that as $h \to 0$ this stencil becomes diagonally dominant.

3.1.4 Two dimensional boundary finite differencing

Our approach to finite differencing in one dimension is general enough that we can apply these principles to multiple dimensions very easily. In Figure 5.1 we show the difference between two types of boundaries. Boundary type one is defined such that there are no grid points defined in two directions. Once such point in the figure is the corner point of the points surrounded by diamonds. Boundary type two has one point undefined in all four directions, which is shown by the center point surrounded by circles. The boundary condition that we will apply is again a general mixed boundary condition

$$\alpha u_x + \beta u_y + \gamma u = g \tag{3.31}$$

Figure 3.6: The discretized domain, where we "straddle" the boundary.



Now we derive a stencil for the equation represented by the point centered by the circles . We can finite difference u_x to second order by applying centered differencing in the x direction, yet we can not apply centered differencing in the y direction. [4] We use the interior condition to solve for the point in the y direction, and then end up with a second order stencil involving just four points. We have problems on boundaries of type 1, since there are two points undefined. For these two points we need two extra equations. One equation will come from the interior equation. The other equation could come from an averaging condition,

$$\frac{1}{2}u[j+1] + \frac{1}{2}u_i[j-1] - u[j] = 0,$$

to obtain another equation but this equation will result in an overall stencil which will be only first order accurate as we have shown previously. A better approach is to apply center differencing in only one of the directions, and apply second order forward differencing in the other direction. We do this by first applying center differencing in the x direction for the diamond points in Figure 5.1 by introducing an extra point, and equation, in the x direction. We then apply forward differencing to second order in the y direction. To extend this principle in three dimensions, we might have to apply center difference in one direction and forward and backward differencing in the other directions. This leads to 9 point stencils in three dimensions.

Our approach in two dimensions has assumed that the discretized boundary points conformed to the physical boundaries. We need to develop a formalism when this is not the case. In the paper by Cook[20] *et al.*, we saw that the Cartesian algorithm had a serious flaw in this regard. It assumed that the physical boundary conformed to the discretized boundary points. We feel that the derivations of the correct equations are much to complex to pursue.

[4] We reproduced this discussion from chapter 5, to aid in the current discussion.

Thus, we feel the best approach is to apply first order finite differencing on the boundaries, and second order finite differencing on the interior, and then use deferred correction on the boundaries to get higher order results [14].

3.2 Basic iterative solvers

In this section we will discuss how to solve large linear algebraic equations which result from finite differencing PDE's. It is our feeling that direct solvers are inferior to iterative solvers for solving most, if not all, physical processes, resulting from elliptic PDE's in multiple dimensions.[5] This is due to to the fact that factoring a matrix, A, arising from a finite difference scheme involves the phenomenon of fill-in. The upper and lower triangular factors will in general have non-zero elements which are zero in the corresponding positions of the original matrix. A lot of work has been devoted to designing algorithms which reduce the fill-in since the operations needed to perform the factorization, as well as the memory storage required, depend on the number of non-zeros in the upper and lower triangular matrices[27][28].

We will typically use direct methods *only* when the iterative methods will not converge. (Sometimes in a multigrid algorithm[6] the equations will not converge. We must therefore use direct techniques to solve the system, which for multigrid techniques, will only be used only very small systems (n is small).)

Since we did not have to use direct methods for the problems in this dissertation, we will only discuss iterative methods in this section. There are many textbooks on iterative methods and we do not attempt to provide a complete coverage of the topic. (See, for example Varga[50] and Young[58]) .[7] We wish only to highlight some areas that we feel will be useful for the reader, and will help in formulating the multigrid approach.

3.2.1 Linear equations

Let

$$Au = f \tag{3.32}$$

denote a system of n linear equations such as those represented by equation (3.14) so that A is a $n \times n$ (generally sparse) matrix.[8] We will typically compare the various iterative methods

[5] Of course, for one dimensional problems, direct solvers are at least as efficient as the best iterative solvers, if not more efficient.

[6] Multigrid is discussed in Chapter 4.

[7] We will follow Choptuik's discussion of iterative methods[15].

[8] A sparse matrix is defined as a matrix which has more zeros than non-zeros in the array elements.

used in this dissertation by examing the number of operations, $N = n^2$, required to solve the system of equations.[9]

In an iterative method for solving (3.32), a series of iterates $u^{(n)}$ is generated, such that

$$\lim_{n\to\infty} u^{(n)} = u \tag{3.33}$$

where we start with some initial estimate, $u^{(0)}$ of the solution. Iterative methods, in principle, require an infinite number of operations to determine u. In practice, the solution is generally terminated when $u^{(n)}$ is "close" to u.

A common method used to monitor the convergence of the solution is to compute at each iteration, the residual vector $r^{(n)}$ defined by

$$r^{(n)} \equiv Au^{(n)} - f. \tag{3.34}$$

Now if the iteration converges, then

$$\lim_{n\to\infty} r^{(n)} = 0. \tag{3.35}$$

We then define the stopping criterion when

$$||r^{(n)}|| \leq \epsilon \tag{3.36}$$

where $||\cdot||$ denotes some discrete norm and ϵ is a convergence criterion, which is usually given before the iteration process begins.

The error vector $e^{(n)}$ of the n^{th} iteration is defined by

$$e^{(n)} \equiv u^{(n)} - u \tag{3.37}$$

Now notice that by the uniqueness of the solution u, $r = 0$ if and only if $e = 0$. But it may not be true that, when the norm of r is small, the norm of e will be too. This is then a measure of the conditioning of the system. [10] From (3.32) and (3.34) we observe the following relation between the error and the residual:

$$Ae = r \tag{3.39}$$

which is known as the residual equation.

---

[9] An operation will typically be a multiplication, division, or addition of two floating point numbers.

[10] The condition number, cond(A), of the matrix, provides a measure of how "close" the matrix is to being numerical singular.

$$\mathrm{cond}\,(A) \equiv ||A||\ \ ||A^{-1}|| \quad 1 \leq \mathrm{cond}\,(A) \leq \infty \tag{3.38}$$

where $||\cdot||$ denotes any vector norm. When cond(A) is very large, A is aid to be ill-conditioned.

Iterative methods are characterized by the manner which the new iterate is calculated from previous iterates. A general class of iterative methods may be defined by[58]

$$u^{(n+1)} = G^{(n)}\left(u^{(n)}, u^{(n-1)}, \cdots, u^{(0)}\right) + c^{(n)}, \tag{3.40}$$

where $G^{(n)}$ is an operator which can change from iteration to iteration as could the vector $c^{(n)}$. We will only deal with methods where $G^{(n)}$ is linear, constant and only operates on the current solution estimate, and where the vector $c^{(n)}$ is also constant. In this case we rewrite the previous equation as

$$u^{(n+1)} = Gu^{(n)} + c \tag{3.41}$$

where G is known as the amplification matrix. We can now see that

$$\begin{aligned} e^{(n)} &= G^{(n)}e^{(0)} \\ &= G^{(n)}\left(u^{(0)} - u\right) \end{aligned} \tag{3.42}$$

where $G^{(n)}$ is the nth power of the amplification matrix. Thus, the iterative procedure will converge if

$$\begin{aligned} \lim_{n\to\infty} \|e^{(n)}\| &= \lim_{n\to\infty} \|G^{(n)}u^{(0)}\| \\ &= 0 \end{aligned} \tag{3.43}$$

To determine whether or not the iterative method will converge, we examine the spectral radius, $\rho(G)$, of G, defined by

$$\rho(G) \equiv \max_i |\lambda_i(G)| \tag{3.44}$$

where $\lambda_j(G)$ are the eigenvalues of G. In order for the method to converge, we must have $|\lambda_i| < 1$ for all i. An important aspect of the spectral radius is that it will tell you how fast the system will converge. In general

$$\lim_{n\to\infty} \frac{\|e^{(k+1)}\|}{e^{(k)}} = \rho(G). \tag{3.45}$$

We can now define an asymptotic rate of convergence, ρ_{R} as

$$\rho_{\mathrm{R}} = \log_{10}\left(\rho(G)^{-1}\right) \tag{3.46}$$

where the reciprocal of ρ_{R} is the number of iterations which must be performed in order to reduce the error by a factor of ten.

Since for any but the very simplest iterative schemes, the explicit form of the amplification matrix becomes very complicated, we will describe the iterative methods in a manner which is closer to the way they are implemented.

We will now discuss two general iterative techniques which will prove useful in studying the multigrid technique in Chapter 4. Both of these methods are used to solve equation (3.32), but we will in general look at the solution of equation (3.7).

3.2.2 Jacobi iteration

The first method that we look at is called the Jacobi iteration, which results from visiting each grid point in succession, changing the value of each unknown so that, using the required neighboring values from the previous iteration, the local difference equations are satisfied. Thus, the Jacobi iteration for the j^{th} equation of (3.7) is defined by

$$u[j]^{(n+1)} = \frac{1}{2}\left(u[j-1]^{(n)} + u[j+1]^{(n)} - h^2 f[j]\right), \qquad 1 \le j \le N-1. \tag{3.47}$$

There is a simple modification which can be made to the Jacobi iteration. As before, we compute the new Jacobi iterate using equation (3.47),(which we will refer to as $u[j]^{(*)}$), where however $u[j]^{(*)}$ is now only an intermediate value. The new approximation is given by the weighted average

$$u[j]^{(n+1)} = (1-\omega)\, u[j]^{(n)} + \omega u[j]^{(*)}, \qquad 1 \le j \le N-1 \tag{3.48}$$

where ω is a real, freely specified weighting factor. This last equation, which defines the weighted Jacobi method can be rewritten in terms of the residual vector, $r^{(n)}$

$$u[j]^{(n+1)} = u[j]^{(n)} - \omega D^{-1} r^{(n)} \tag{3.49}$$

Here D is the main diagonal of L, so $D^{-1} = Ih^2/2$, where I is the $n \times n$ identity matrix. The eigenvalues of the weighted Jacobi method, are well documented [10][12] and are given by

$$\lambda_j\left(G_\omega\right) = 1 - 2\omega \sin^2\left(\frac{j\pi}{2N}\right), \qquad 1 \le j \le N-1 \tag{3.50}$$

The modes in the lower half of the spectrum, with wavenumbers in the range $1 \le k \le N/2$, are low-frequency or smooth modes. The modes in the upper half of the spectrum, will be called high-frequency or oscillatory modes.

If we now replace N by h^{-1} (since $Nh = 1$) in the above equation, and expand the sin term to $O\left(h^2\right)$ we determine that the eigenvalue associated with the smoothest mode, (*i.e.* when $j = 1$) , is

$$\lambda_1 = 1 - 2\omega \sin^2\left(\frac{\pi h}{2}\right) = 1 - \frac{\omega \pi^2 h^2}{2} + O\left(h^4\right) = 1 + O\left(h^2\right)$$

Clearly, for small h this eigenvalue is approximately 1, and therefore, the convergence of the iteration will be very slow. On the other hand, for the high-frequency mode corresponding to $j = N-1$, we have

$$\lambda_{N-1} = 1 - 2\omega \sin^2\left(\frac{\pi\left(N-1\right)}{2N}\right) \approx 1 - \frac{\omega \pi^2}{2}$$

which is clearly much less than 1.[11] Thus, high frequency components are damped much faster than the low frequency components, and this is a property shared by many iterative schemes. In the next chapter we will study these eigenvalues further, and in Chapter 6, for pedagogical purposes, we shall briefly discuss the application of the Jacobi method to a model problem.

Besides the fact that the Jacobi method is an extremely poor solver for elliptic equation, we see that it also requires storage of the current iteration variables, as well as storage for the previous iteration. We see that this drawback can be overcome by the Gauss-Seidel Method: if the Jacobi method converges, then, in general, $u^{(n+1)}$ will be a better estimate of u than $u^{(n)}$. This suggests that an improvement on the Jacobi iteration might result by using newly calculated quantities whenever possible in the course of an iteration.

3.2.3 Gauss-Seidel and SOR iteration

Perhaps a more familiar iteration method is the Gauss-Seidel method. This method may be viewed as a simple modification of the Jacobi method whereby most-recently-computed components, $u[j]^{(n+1)}$, are used whenever possible. This is an intuitive modification, since if the Jacobi method converges, then $u[j]^{(n+1)}$ will be a better estimate of $u[j]$ than $u[j]^{(n)}$. As just mentioned, this also reduces memory requirements since during the Jacobi iteration we must provide storage for all of the $u[j]^{(n)}$ and $u[j]^{(n+1)}$ (two vectors of length N) whereas for the Gauss-Seidel we "overwrite" $u[j]^{(n)}$ with $u[j]^{(n+1)}$. Up to now we have assumed that the components of u are updated in ascending order, commonly known as lexicographic order. For the weighted Jacobi method, the order does not matter since components are never over-written. However, for the Gauss-Seidel method, the order of updating is significant. Instead of sweeping through the components in ascending order, we could sweep through the unknowns using the reverse order or we could even alternate between the two, which is known as symmetric Gauss-Seidel. Another very important method results from first updating all the even components, and then updating all the odd components. This ordering, Red-Black, gets it's name from two dimensional applications. One significance of the ordering is for their implementation on a parallel or vector computer. The Red-Black ordering allows the method to be highly vectorizable, which is not true about lexicographical ordering. There are other ramifications of using Red-Black order, particularly for multi-grid.

For our model problem, equation (3.7), the Gauss-Seidel iteration can be expressed as

$$u[j]^{(n+1)} = \frac{1}{2}\left(u^{(n+1)}[j-1] + u^{(n)}[j+1] - h^2 f[j]\right), \quad 1 \le j \le N-1 \tag{3.51}$$

[11] Unless ω was extremely small. Since if ω is small then clearly convergence must be small at all wavelengths.

Let us now look at the eigenvalue structure of the amplification matrix for the Gauss-Seidel method. Let us assume that the system we are going to look at is defined by the finite difference equations given in equation (3.6) for the interior points, and that there are a total of five points in the domain, three interior points, and two boundary points. Let us also assume that we apply boundary conditions:

$$\begin{aligned} \alpha u_x + u = g|_{x=0} \\ u(x_N) = u[N] \end{aligned} \tag{3.52}$$

The equation at $x = 0$ is then finite differenced to

$$\epsilon u[1] + (1-\epsilon)\, u[0] = g \tag{3.53}$$

where $\epsilon \equiv \alpha/h$. Once again we write our system of equations as

$$Au = f$$

where now A is given by

$$A = \begin{pmatrix} 1-\epsilon & \epsilon & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}. \tag{3.54}$$

We now break A into upper– and lower–triangular matrices U and L:

$$U = \begin{pmatrix} 0 & \epsilon & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

$$L = \begin{pmatrix} 1-\epsilon & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 \\ 0 & -1 & 2 & 0 \\ 0 & 0 & -1 & 2 \end{pmatrix}.$$

so that $A = L + U$. A particular Gauss-Seidel iteration for our equations is the given by:

$$Lu^{n+1} + Uu^{(n)} = f \tag{3.55}$$

or

$$u^{(n+1)} = L^{-1}\left(f - Uu^{(n)}\right) \tag{3.56}$$

The best way to look at the convergence properties of iterative solvers is to look at the amplification matrix defined from the residual equation. So,

$$r^{(n+1)} = (L+U)\, u^{(n+1)} - f$$

$$
\begin{aligned}
&= (L+U)\,L^{-1}\left(f - U u^{(n)}\right) - f \\
&= f - U u^{(n)} + U L^{-1} f - U L^{-1} U u^{(n)} - f \\
&= U L^{-1}\left(f - U u^{(n)} - L u^{(n)}\right) \\
&= U L^{-1}\left(f - (L+U)\,u^{(n)}\right) \\
&= -U L^{-1} r^{(n)}
\end{aligned}
\tag{3.57}
$$

Thus,

$$
r^{(n+1)} = G\,r^{(n)} \tag{3.58}
$$

where $G \equiv -U L^{-1}$ is the amplification matrix (but different from G defined previously. In particular, for our model problem we have

$$
G = \begin{pmatrix} -\frac{1}{2}\frac{\epsilon}{\epsilon-1} & \frac{\epsilon}{2} & 0 & 0 \\ \frac{1}{4}\frac{1}{\epsilon-1} & \frac{-1}{4} & \frac{1}{2} & 0 \\ \frac{1}{8}\frac{1}{\epsilon-1} & \frac{-1}{8} & \frac{-1}{4} & -\frac{1}{2} \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.59}
$$

Now, the eigenvalues are

$$
0 \quad ,0 \quad ,\frac{1}{2}\frac{8\epsilon - 4 + 4\sqrt{2\epsilon^2 - 2\epsilon + 1}}{8\epsilon - 8}, \frac{1}{2}\frac{8\epsilon - 4 - 4\sqrt{2\epsilon^2 - 2\epsilon + 1}}{8\epsilon - 8} \tag{3.60}
$$

We see that when once ϵ becomes a little greater than (1 /2), the spectral radius, $\rho(G)$, is greater than 1 which means that the system will not converge. We also see that as ϵ becomes larger than zero, the eigenvalues will move closer to one. This signifies that the Gauss-Seidel iteration will take a much longer time than if $\epsilon = 0$.

Because the above methods solve for one new function value at a time, they are often referred to as point-relaxation methods. Another class of relaxation methods involves the simultaneous update of a group of unknowns. For example, if we were working in two dimensions and divided up the y axis into M lines, one could then solve the system of equations (3.51) simultaneously for all the points that are on the same line. We can then extend the red-black idea to lines, resulting in so-called "zebra" solvers.

Researchers[58] who used the Gauss-Seidel method for solving systems of difference equations discovered that convergence could be accelerated by modifying the iteration so that

$$
u[j]^{(n+1)} = \omega\ u[j]^{\star} + (1-\omega)\,u[j]^{(n)} \tag{3.61}
$$

where $u[j]^{\star}$ is determined by the right hand side of (3.51), and ω is called the relaxation parameter. If $0 < \omega < 2$, then the above defines the well-known successive over-relaxation (SOR) iteration which incorporates the Gauss-Seidel iteration as a special case when $\omega = 1$.

Typically, for large elliptic systems the S.O.R. method requires fewer operations than the previous methods [58].

In general, it is quite difficult to get an optimal value of ω, so one must do some numerical experimentation or use some sort of adaptive procedure which attempts to estimate the optimal ω in the course of the solution process.

3.2.4 Interpolation, extrapolation and determination of derivatives to high order

We will see in the next chapter, that in order to use deferred correction, we will need to determine approximate values for a function and its derivatives at spatial locations not coincident grid points. We use polynomial interpolation for their evaluation. We determine the function values to be interpolated, along with their derivatives via straightforward Taylor series expansion. This method is not unique in any sense.

Since we deal with lattices, and the finite difference equations were derived from truncated Taylor series expansions, it becomes natural to think of the interpolation as just using another Taylor expansion to determine the value of the function. Let's suppose our computational domain is again shown in Figure 3.6, and we wish to determine the first derivative to fourth order, and the function value at this boundary point to fourth or higher order.[12] We may at first write down the Taylor expansion for the point u_b which would make the first derivative fourth order accurate, *i.e.*, we write

$$\begin{aligned} u_b &= u\left(a-(h-\xi)\right) \\ &= u+(\xi-h)\,u_x+\frac{1}{2}(\xi-h)^2\,u_{xx}+\frac{1}{6}(\xi-h)^3\,u_{xxx} \\ &+ \frac{1}{24}(\xi-h)^4\,u_{xxxx}+O\left(h^5\right), \end{aligned} \tag{3.62}$$

which shows that the first derivative is determined up to fourth order,

$$\begin{aligned} u_x &= \frac{1}{\xi-h}\Big(u_b-u-\frac{1}{2}(\xi-h)^2\,u_{xx} \\ &- \frac{1}{6}(\xi-h)^3\,u_{xxx}-\frac{1}{24}(\xi-h)^4\,u_{xxxx}\Big)+O\left(h^4\right) \end{aligned} \tag{3.63}$$

There are 5 points which we need in order to determine the first derivative to fourth order. Our basic algorithm to find the first derivative up to fourth order, and the function interpolated value, is broken up into 2 parts. Our algorithm finds the matrix values used in the Taylor expansions and then performs a LUD decomposition[13]. This routine is called only

[12] We use this derivation in Section 6.5.
[13] LUD decomposition is described in Numerical Recipes[41].

in the beginning of the programs since it is a $O\left(N^3\right)$ operation [41]. The next part of the algorithm performs the backwards substitution when the function and/or the derivative is needed to be determined. This is an order $O\left(N^2\right)$ operation.

For multiple dimensions we use the same algorithm although the number of terms in the matrix is greatly increased. For example, the expansion used in three dimensions is

$$\begin{aligned} f(x+\alpha, y+\beta, z+\gamma) &= f + \alpha f_x + \beta f_y + \gamma f_z \\ &+\frac{1}{2}\left(\alpha^2 f_{xx} + \beta^2 f_{yy} + \gamma^2 f_{zz}\right) + \alpha\beta f_{xy} + \alpha\gamma f_{xz} + \beta\gamma f_{yz} \\ &+\frac{1}{6}\left(\alpha^3 f_{xxx} + \beta^3 f_{yyy} + \gamma^3 f_{zzzz}\right) + \frac{1}{2}\Big(\alpha^2\beta f_{xxy} + \alpha^2\gamma f_{xxz} + \alpha\gamma^2 f_{xzz} \\ &+\alpha\beta^2 f_{xyy} + \beta^2\gamma f_{yyz} + \beta\gamma^2 f_{yzz}\Big) + \alpha\beta\gamma f_{xyz} + \frac{1}{24}\Big(\alpha^4 f_{xxxx} + \beta^4 f_{yyyy} \\ &+\gamma^4 f_{zzzz}\Big) + \frac{1}{6}\Big(\alpha^3\beta f_{xxxy} + \alpha^3\gamma f_{xxxz} + \alpha\beta^3 f_{xyyy} + \beta^3\gamma f_{yyyz} + \alpha\gamma^3 f_{xzzz} \\ &+\beta\gamma^3 f_{yzzz}\Big) + \frac{1}{4}\Big(\alpha^2\beta^2 f_{xxyy} + \alpha^2\gamma^2 f_{xxzz} \\ &+\beta^2\gamma^2 f_{yyzz}\Big) + \frac{1}{2}\Big(\alpha^2\beta\gamma f_{xxyz} + \alpha\beta^2\gamma f_{xyyz} + \alpha\beta\gamma^2 f_{xyzz}\Big) \end{aligned} \tag{3.64}$$

which uses 35 points in the stencil!

Now extrapolation involves the same principles as interpolation. We only extrapolate for points which are $O(h)$ from the actual computational domain.

We have seen most of the basic components of the numerical methods that we use in this dissertation. In the next chapter, we will use some of these components to describe the MLAT method as well with deferred correction and Richardson extrapolation.

Chapter 4

Multi level adaptive techniques

The main purpose of this chapter is to provide a brief introduction to multigrid methods and the essential ingredients used in their development. This chapter will also introduce the standard multigrid notation, which will be widely used in later chapters. We also introduce the techniques of Richardson extrapolation, deferred correction and AMR since we feel these techniques work extremely well in the context of a multigrid algorithm. We in no sense give a complete overview of multigrid, for we feel there are many excellent papers on this subject. Good sources of reference include Brandt's excellent 1977 paper[8], Brandt's Guide to multigrid development[9], along with Briggs' multigrid tutorial[10] and Choptuik and Unruh's paper in General Relativity and Gravitation[12].

4.1 Philosophy

The basic idea of multigrid is to work with a sequence of grids, where each coarser grid helps accelerate the convergence of the finest grid. This is only half the picture for Multi Level Adaptive Techniques, MLAT: the other half is that we use the solution of a coarse grid problem as an initial guess of the fine grid problem. Multigrid algorithms use a series of discretization levels $l = 0, \ldots, l_{\max}$ where $l = 0$ is the coarsest level, and $l_{\max}$ is the finest and $h_l = 2h_{l+1}$.

This is only half of the picture for Multi Level Adaptive Techniques, where the other half is that we use finer discretizations only where they are needed.

Our basic philosophy is to follow Brandt's golden rule[9]

> **The amount of computational work should be proportional to the amount of real physical changes in the computed systems.**

Thus, algorithms that require more than $O(N)$ operations to solve a discrete system of $O(N)$ equations are working too hard to extract the physics of the problem, and therefore should be discouraged. We use multigrid for three reasons. The first reason is that it does obey

Brandt's golden rule, since it is a $O(N)$ solver. The second reason we use multigrid is that it provides us with a natural way of incorporating adaptive meshes along with many other advanced techniques, such as deferred correction and Richardson extrapolation. The final reason is that multigrid methods provide natural stopping criterion used for determining when a problem should be considered solved. The other "classical" methods such as S.O.R. are neither $O(N)$ solvers, nor do they allow adaptive meshes to be easily incorporated. These methods do not even provide any natural stopping criteria; the stopping criteria are preset in the algorithm to some value which does not change.

Our other stipulation is that we want high order accuracy at the lowest possible cost. We will see later that Richardson extrapolation helps accomplish this task rather nicely. But in order to use Richardson extrapolation, the solution must behave nicely, *i.e.*, the function must be smooth everywhere, including the boundaries. This will be true in the physics of black holes, since we do not anticipate the development of discontinuities (shocks) in our solutions. In fact, Cook[20] has already been able to extrapolate the solution of the initial value problem.

Before we begin to describe each part of the multigrid algorithm, we present a basic sketch of the multigrid algorithm. We start with a fine grid and a coarse grid. The residuals on the fine grid are then smoothed until all of the high frequency components are removed. Their residuals are then transferred to the coarse grid. The coarse grid equations are then modified by these residuals. The coarse grid equations are then solved, and subsequently transferred to the fine grid. The fine grid then modifies its solution by the correction which was found from the coarse grid equation. Multigrid then recursively applies these ideas to a series of grids, where the coarse grid equations are smoothed on the intermediate grids, and solved only on the coarsest grid. Thus, there are four major components of a multigrid routine. These components are smoothing, transfers from a fine grid to a coarse grid, transfers from a coarse grid to a fine grid, and the solution of the coarsest grid .

4.2 Smoothing

Perhaps the most important part of any multigrid algorithm is the smoothing of the residuals. Multigrid routines transfer the residuals from a fine grid to a coarse grid, which has a grid spacing equal to twice that of the fine grid. High frequency components in the residuals on the fine grid can not be accurately represented on the coarse grid. Thus, the residuals must be smooth in order to approximate them on the coarse grid. If a multigrid algorithm is working properly with a reasonably efficient smoother then over 70 % of the computational time will be spent on smoothing[9]. This means that not only do we want a good smoother, but we also want a fast smoother. For example, if lexicographic Gauss-Seidel was used for smoothing, we would see that this smoother would operate very slowly on a vector or parallel computer, whereas if a well coded Red-Black Gauss-Seidel smoother was used for the smoothing, the speed of this routine would greatly increase.

All of the smoothers used in this dissertation utilize Red-Black Gauss-Seidel, but we feel that it is helpful to describe Jacobi smoothing too. We note that as various researchers have shown [15][8][10] , Jacobi is not is good as a smoother as Gauss-Seidel.

4.2.1 Damped Jacobi smoothing

The Jacobi method allows us to use analytic methods to examine its smoothing properties. We examine the smoothing behavior of

$$\frac{\partial^2 u(x)}{\partial x^2} = f(x) \quad x \in [0,1], \tag{4.1}$$

where $u(x) = \sin(x)$, and $f(x) = -\sin(x)$. In chapter 3 we described how the eigenvalues behave for high and low frequency. The most interesting property of this method is its smoothing. The high frequency components of the eigenvalues are quickly damped; meaning a highly oscillatory function will be smoothed out in roughly a few iterations. Figure 4.1 shows the smoothing behavior of the damped Jacobi algorithm. Here we took $\omega = 2/3$, where ω was defined by equation (3.48). The initial guess of u was taken to be

$$u[j] = -1^j \quad j = 1 \cdots n \tag{4.2}$$

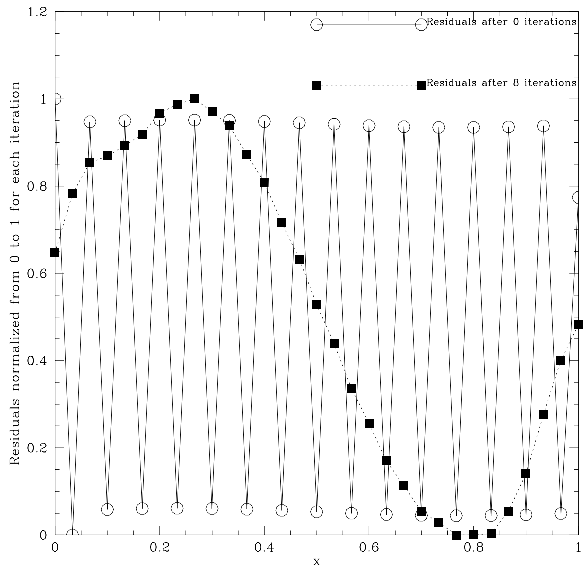
which clearly has high frequency components. To show the smoothing properties of the Jacobi iteration, we plot residuals for the 0^{th} and 8^{th} iteration in Figure 4.1. After 8 iterations we see that the high frequency components were effectively eliminated. Although Jacobi is a good smoother, previous work by various authors [12][10] shows that Red-Black Gauss-Seidel is a superior smoother so we shall look at this technique next.

4.2.2 Red-Black Gauss-Seidel smoothing

We use Red-Black Gauss-Seidel($\omega = 1$) smoothing since we find it to be the best smoother for our purposes, and since it is well documented[10] [12] that this is an excellent smoother. The only thing which we will point out in this section is the reason to use Red Black Gauss-Seidel, rather than S.O.R., for smoothing.

The S.O.R. iteration converges faster than the Gauss-Seidel iteration, so we would like to see how the "ω" parameter in S.O.R. affects the smoothing rate. Again we analyze the Jacobi example, equation (4.2.1). Our stopping criterion for this experiment is when the norm of the residual is reduced by one order of magnitude from its initial value. In this

Figure 4.1: A graphical example of Jacobi smoothing.



experiment, we let $N = 65$, which means that the optimal value for ω should be

$$\omega_{\text{optimal}} = \frac{2}{1 + \sin \frac{\pi}{2N}} \approx 1.95 \tag{4.3}$$

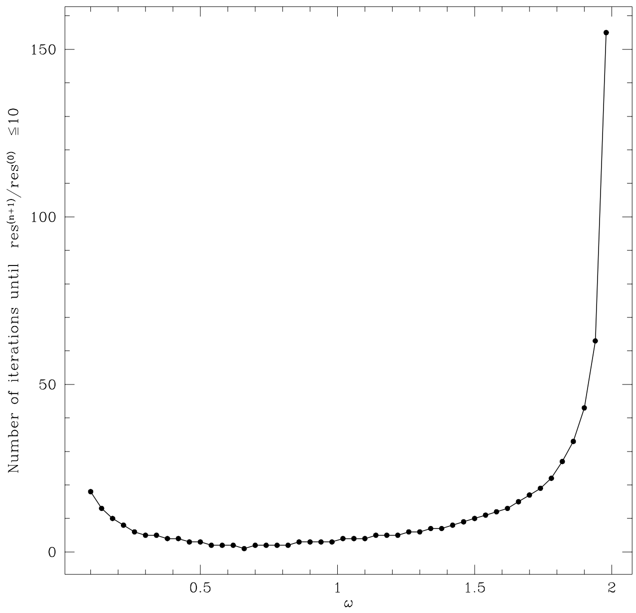
Figure 4.2 clearly shows that this value of ω takes far more iterations than when $\omega = 1$. Thus, we shall always let $\omega = 1$ for smoothing the interior points. When we must solve a problem on a coarse grid, we use ω_{optimal} as our over-relaxation parameter.

4.2.3 Smoothing boundaries

Since the boundary region is of lower dimension than the interior we can generally allow more work per grid point here without seriously degrading the overall performance of the algorithm. In multigrid programs, it is usually good practice to treat boundary equations separately from interior equations[9]. Thus the usual method for handling the boundary is to smooth the boundary equations independently of the interior equations.

For one dimensional smoothing Brandt[9] says that the boundary conditions need not be relaxed at all since the errors there are not functions that can be smoothed in any way. In general we agree with this statement when it is applied to simple problems. In some of our test problems in one dimension, we allowed partial relaxation on the boundary point.

Figure 4.2: An example of S.O.R. smoothing.



Thus, for the boundary problem

$$Bu = g$$

we smooth this as

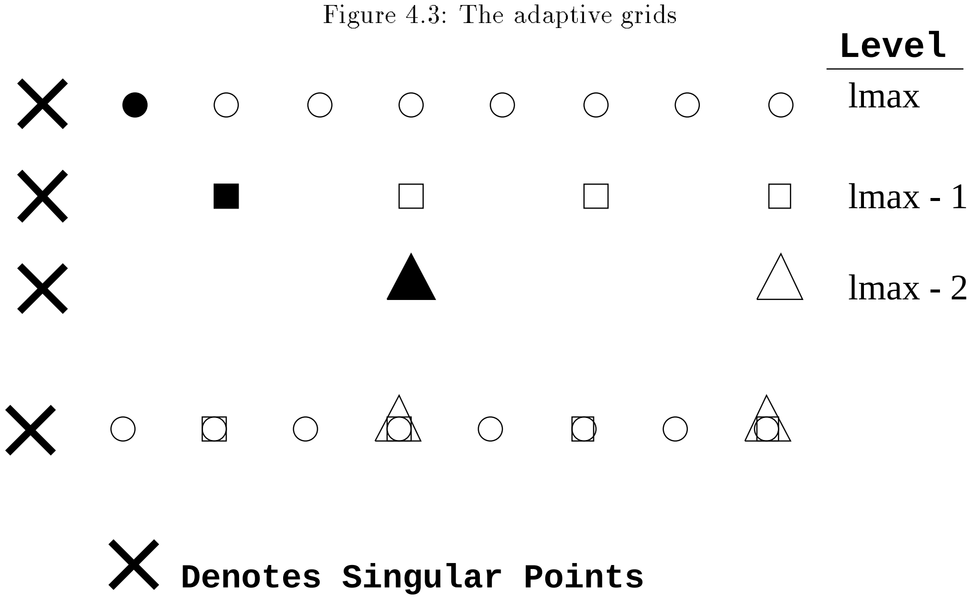
$$u^{(n+1)} := u^{(n)} + \alpha \left(Bu^{(n)} - g - u^{(n)} \right) \tag{4.4}$$

where α is the smoothing parameter. Now Brandt's statement would put $\alpha = 0$, yet we have verified[1] that there are instances where α should be equal to one. One such instance is when we have a grid structure that violates the Berger and Oliger [2] principle that any grid at level $l+1$ (fine level) should be *properly contained* by a parental grid at level l. This is shown in figure(4.3), where the finest grid contains points which are outside of the domain of the coarser grids. We term these points as being "not properly contained". In general, we find that for uniform grids the fastest convergence rates occur when $\alpha \approx 0$. Usually (and in accord with Brandt's general observations), we believe that α should be chosen to result in the smoothest interior residuals.

In multi-dimensional boundary smoothing we must obey the principle that we only smooth the residuals along the boundary. The residuals to the boundary equation must be

[1] See Section 6.5

Figure 4.3: The adaptive grids



smoothed independently of the values on the interior points. We must again demand that the smoothing on the boundaries not disrupt the smoothness of the residuals associated with the interior equations. Brandt makes several suggestions to accomplish this. One suggestion is to apply the smoothing operator

$$\frac{\partial^2}{\partial s^2} Bu = \frac{\partial^2}{\partial s^2} g \tag{4.5}$$

instead of $Bu = g$ on the boundary, where s is the boundary arclength. Brandt also suggests that when the boundary smoothing factor is not as good as the interior one, a few boundary sweeps can be performed for each interior sweep.

For geometries such as in Figure 4.4, we employ Brandt's suggestion of using a couple of relaxation sweeps per interior sweep and also a suggestion of Brandt's[8] average the residuals along the boundary, *i.e.*

$$r[l] = \frac{1}{2}\left(r[l-1] + r[l+1]\right). \tag{4.6}$$

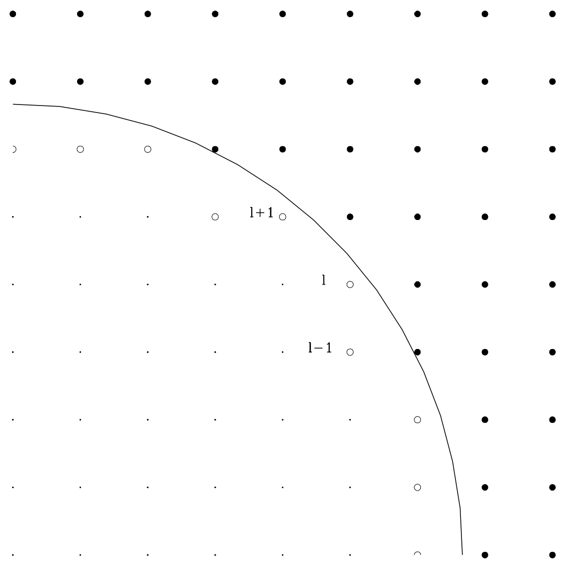
Here we treat the residual in this two dimensional problem as a one dimensional object. We equate the point $[l]$ on the boundary with the point at $[j,k]$ in the problem. If we wanted to smooth the boundary at the point $[l] = [j,k]$, using the residuals at $[l-1] = [j,k-1]$ and $[l+1] = [j-1,k+1]$, (as shown in Figure 4.4, for the then boundary equation,

$$\alpha u + \beta u_\rho = g$$

where

$$\frac{\partial u}{\partial \rho} = \frac{x}{\rho}\frac{\partial u}{\partial x} + \frac{y}{\rho}\frac{\partial u}{\partial y} \tag{4.7}$$

Figure 4.4: A 2D domain, where the points "inside" the inner boundary are not in the computational domain.



with $\rho = \sqrt{x^2+y^2}$, we have

$$\begin{aligned}\alpha u[j,k] &+ \beta\frac{x[j]}{\rho}\frac{u[j+1,k]-u[j,k]}{h}+\beta\frac{y[k]}{\rho}\frac{u[j,k+1]-u[j,k]}{h}-g[j,k]\\ &= \frac{1}{2}\left(r[j,k-1]+r[j-1,k+1]\right)\end{aligned} \tag{4.8}$$

4.3 Prolongation and restriction operators for interior equations

There are two basic types of transfer operators needed, restriction and prolongation which are denoted by I_h^H and I_H^h respectively. The purpose of the I_h^H is to take grid functions from levels $l \rightarrow l-1$, and the purpose of I_H^h is to take grid functions from levels $l \rightarrow l+1$.

Restriction operators use weighted averages from the fine grid functions to determine the coarse grid functions. Prolongation operators generally use polynomial interpolation to determine the fine grid function.

A simple one dimensional domain is shown in Figure 4.5 for two levels, with $H = 2h$

and $2J-1=j$. The restriction operator is defined in stencil notation as

$$u^H[J] = I_h^H u^h[j] = \left(\frac{1}{4}\ \frac{1}{2}\ \frac{1}{4}\right)\begin{pmatrix} u_{j-1}^h \\ u_j^h \\ u_{j+1}^h \end{pmatrix}, \quad J = 2, N-1, \tag{4.9}$$

which is a weighted average. Another common restriction operator is the injection operator defined as

$$u^H[J] = I_h^H u^h[j] = u^h[j] \tag{4.10}$$

We have empirically found that for all of the problems that we have dealt with, that the use of injections for restrictions tends to make the multigrid routines converge more slowly. However, for one dimensional problems, we use injections for the restriction operator of the boundary values.

Prolongation in one dimension is typically accomplished via polynomial interpolation. For example, linear (accurate to $O\left(h^2\right)$) prolongation/interpolation is defined by:

$$\begin{aligned} u^h[j] &= I_H^h u^H[J] \\ &= \left(\frac{1}{2}\ 0\ \frac{1}{2}\right)\begin{pmatrix} u^H[J-1] \\ u^H[J] \\ u^H[J+1] \end{pmatrix}, \quad j \neq 2J-1 \\ &= (0\ 1\ 0)\begin{pmatrix} u^H[J-1] \\ u^H[J] \\ u^H[J+1] \end{pmatrix}, \quad j = 2J-1 \end{aligned} \tag{4.11}$$

Now two dimensional restriction is more complicated since at various times we use full weighting, half weighting or injections defined as follows (where $j = 2J-1$ and $k = 2K-1$):

Full Weighted Restriction

$$\begin{aligned} u^H[J,K] = I_h^H u^h[j,k] &= \frac{1}{16}\Big(u^h[j-1,k+1] + u^h[j+1,k+1] \\ &+ u^h[j-1,k-1] + u^h[j+1,k-1]\Big) \\ &+ \frac{1}{8}\Big(u^h[j-1,k] + u^h[j+1,k] \\ &+ u^h[j,k-1] + u^h[j,k+1]\Big) \\ &+ \frac{1}{4}u^h[i,j] \end{aligned} \tag{4.12}$$

Half Weighted Restriction

$$\begin{aligned} u^H[J,K] &= I_h^H u^h[j,k] \\ &= \frac{1}{8}\left(u^h[j-1,k] + u^h[j+1,k] + u^h[j,k-1] + u^h[j,k+1]\right) \\ &+ \frac{1}{2}u^h[i,j] \end{aligned} \tag{4.13}$$

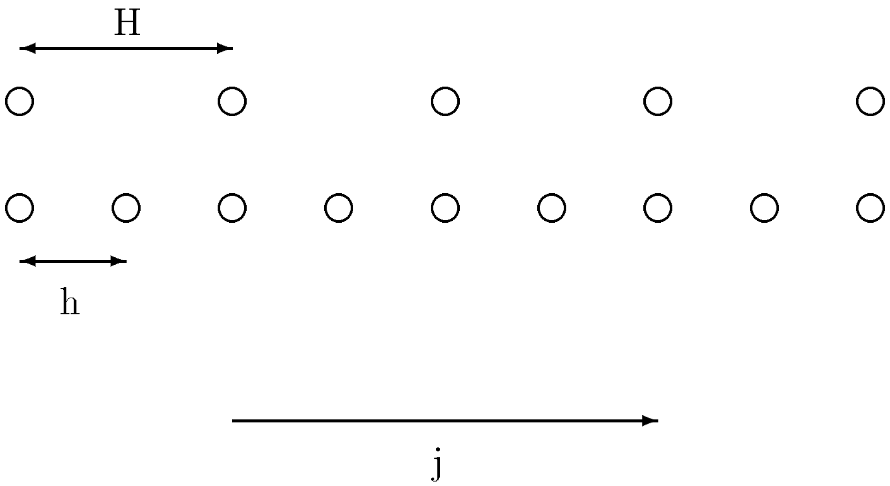
Injection

$$u^H[J,K] = I_h^H u^h[j,k] = u^h[j,k] \tag{4.14}$$

Our most commonly used form of the prolongation operator, I_H^h, in two dimensions is bi-linear interpolation, defined by :

$$\begin{aligned}
u^h[j,k] &= u^H[J,K] \\
u^h[j+1,k] &= \frac{1}{2}\left(u^H[J,K] + u^H[J+1,K]\right) \\
u^h[j,k+1] &= \frac{1}{2}\left(u^H[J,K] + u^H[J,K+1]\right) \\
u^h[j+1,k+1] &= \frac{1}{4}\left(u^H[J,K] + u^H[J+1,K]\right) \\
&+ \frac{1}{4}\left(u^H[J,K+1] + u^H[J+1,K+1]\right)
\end{aligned} \tag{4.15}$$

Figure 4.5: Multiple levels used in multigrid algorithms



4.4 Boundary transfers

The hardest part of a "non-trivial" multigrid program is transferring the function values defined by the boundary equations. Sometimes we must use separate operators for transferring the grid function $\tilde{u}^h$ and the residuals, $\tilde{r}^h$. The boundary transfers are unique to each individual problem so we will come back to this issue in our treatment of example problems in Sections 6.3- 6.7.

4.5 Linear Correction Scheme (LCS) for two levels

The Linear Correction Scheme (LCS) can only be used for linear partial differential equations[8][12][9]. Later we will look at the Full Approximation Storage scheme (FAS)[9][8] [12], which is applicable to non-linear PDE's. Here we are trying to solve the linear set of equations

$$\begin{aligned} L[u(x)] &= f(x) \quad x \in \Omega \\ B[u(x)] &= g(x) \quad x \in \partial\Omega \end{aligned} \tag{4.16}$$

where L and B are some linear differential operators, x is some d-dimensional set of coordinates, Ω is some subset of R^d and $\partial\Omega$ is the $d-1$-dimensional boundary. We finite difference these equations as,

$$\begin{aligned} L^h u^h &= f^h \\ B^h u^h &= g^h. \end{aligned} \tag{4.17}$$

From here on, we will exclude explicit treatment of the boundary equations from our discussion since the techniques involved are essentially the same as those applied to the interior equations.

As in Chapter 3, we define a residual, r^h associated with the current approximation, $\tilde{u}^h$:

$$r^h \equiv L^h \tilde{u}^h - f^h \tag{4.18}$$

The next assumption in the derivation of the Coarse Grid Correction (CGC) algorithm is that r^h is smooth. We presume this will have been accomplished via application of a smoothing iteration as previously discussed.

Since L^h is linear and r^h is smooth we can look for a correction v^h such that

$$u^h = \tilde{u}^h + v^h. \tag{4.19}$$

Thus we have have

$$L^h \left(\tilde{u}^h + v^h\right) = f^h, \tag{4.20}$$

and since L^h is linear,

$$L^h v^h = L^h u^h - L^h \tilde{u}^h$$

Now, substituting the definition of the residual, equation (4.18), into this equation we have

$$L^h v^h = -r^h. \tag{4.21}$$

Since we assume that r^h is smooth we can sensibly pose a version of this last equation on a grid which has a coarser scale of discretization. In particular, we can use a grid with mesh spacing $H = 2h$. Thus the coarse grid equation we wish to solve is:

$$L^H v^H = I_h^H \left(-r^h\right) \tag{4.22}$$

where I_h^H is the fine to coarse grid transfer operator (restriction operator). In anticipation for non-linear operators, we see that we can rewrite this equation in three equivalent forms, namely

$$\begin{aligned} L^H v^H &= I_h^H \left(-r^h\right) \\ L^H \left(u^H - I_h^H \tilde{u}^h\right) &= I_h^H \left(-r^h\right) \\ L^H u^H - L^H I_h^H \tilde{u}^h &= I_h^H \left(-r^h\right) \end{aligned}$$

Equation (4.19) defines the relationship between $\tilde{u}^h$ and v^h. Thus, after solving the coarse grid equation (4.22), we update $\tilde{u}^h$ via

$$\tilde{u}^h := \tilde{u}^h + I_H^h v^H \tag{4.23}$$

where I_H^h is a coarse to fine grid transfer, (prolongation operator).

The correction algorithm is shown in Figure 4.6 for a two level scheme. Here `pre` is the number of times the algorithm will smooth the residuals on a grid before a coarse grid correction takes place, and `pst` is the number of times the algorithm will smooth the residuals after a coarse grid correction. We usually set both of these to two for two dimensional runs, one for one dimensional runs, and for three dimensional runs, we usually set these to three. The total number of coarse grid corrections is stored in the variable, σ. As Brandt suggests in his Guide to Multigrid Development[9], it is best to start with a two level algorithm when designing a multigrid code. One can typically see problems with the code in the two level scheme much more easily than with multiple levels.

Since the majority of the time in a multigrid code is typically spent doing relaxation sweeps, we can analyze multigrid codes in terms of work units. One work unit, W, is defined to be the work required to perform one full relaxation sweep across the finest grid. The actual work then is directly proportional to the work unit multiplied by the number of points of the finest level. Using this definition, we recall that S.O.R. codes typically require $O(N)$ work units to solve the "model" problem. Multigrid algorithms generally compute a solution with $O(1)$ work units. That is, the work per grid point tends to be constant which is the best we can hope for.

We define the convergence rate as

$$\rho \equiv W^{-1} \log_{10} \frac{||r^{\text{initial}}||}{||r^{\text{final}}||} \tag{4.24}$$

Figure 4.6: The algorithm for the two level LCS.

```
PROCEDURE TWO_LEVEL_CS_SCHEME
   Initialize_grid_variables
   DO p = 1 ··· pre
      RELAX    Smooth the residual
      r = ℓ1 (r^h)
   END DO    for p
   WHILE r ≥ ε
      r^H := I_h^H r^h
      SOLVE( L^H v^H = I_h^H (-r^h) )    solve the coarse grid equation
      v^h := I_H^h v^H    Prolongate the correction
      ũ^h := ũ^h + v^h    update u^h
      DO q = 1 ··· pst
         RELAX Smooth the residual
      END DO    for q
   END WHILE    for r
END PROCEDURE .
```

We see ρ^{-1} is the number of sweeps needed to reduce the residuals by one order of magnitude.

4.6 Linear correction scheme for multiple levels

The primary difference between using multiple levels and using just 2 levels is that we apply the linear correction scheme recursively, completely solving only on the coarsest level. Now instead of calling the right-hand side of the residual equation $-r^H$, we will simply call it f^H since it is just another right-hand side. Before we can give a general scheme, we must point out that there are various ways to cycle through the grids. The simplest cycle is the V-cycle shown by Figure 4.9 and defined by Figure 4.7, where $l_{\max}$ is the maximum level and l_{coarse} is the coarsest level.

Now one could imagine that instead of correcting u^h from the time we solve on the coarsest grid down to the finest level, we could perform what is known as a W-cycle as shown in Figure 4.10. We will not discuss the more general μ cycle described in Briggs[10]. In general our solvers have the capability of using μ cycles but we usually let $\mu = 1$ or 2, which corresponds to the V and W cycles.

Recall in section 4.1 that one of the original ideas behind multigrid was that if we wanted to solve the problem $L^h u^h = f^h$, we could solve the problem $L^H u^H = f^H$, and then interpolate u^H to u^h to use as an initial guess for the solution of $L^h u^h = f^h$. We use the high order polynomial interpolation operator[9]

$$\tilde{u}^h := \Pi_H^h u^H \tag{4.25}$$

Figure 4.7: The algorithm for the multiple level, V-cycle, LCS scheme.

```
PROCEDURE MULTIPLE_LEVEL_CS_SCHEME( l_max, ncyc, pre, pst )
   Initialize Variables
   Initialize Memory
   DOcycle= 1 , ncyc
      VCYCLE( cycle, l_max , pre, pst)
   END DO
END PROCEDURE

PROCEDURE VCYCLE( cycle, l_max, pre, pst )
    Cycle up to the coarsest level
    DOl = l_max , l_coarse+1
      IF (l ≠ l_max) OR (cycle = 1) THEN
         DOp = 1 , pre
            RELAX    Smooth the residual
         END DO
      END IF
         f^H = I_h^H r^h
    END DO
   Now solve the system on the coarsest level
    SOLVE(L^H u^H = f^H)
   Now come back down to the finest level, performing the needed corrections
   DOl = l_coarse, l_max − 1
       Determine the correction
       u^h := u^h + I_H^h u^H
       Now smooth out u^h
      DOq = 1 , pst
         RELAX    Smooth the residual
      END DO
    END DO
 END PROCEDURE
```

to provide an initial estimate for the function on the next grid. In practice, Π_H^h usually performs polynomial interpolation, but certain guidelines are set[9]. Thus, we could solve the coarsest problem, use the solution of this as the initial guess for the next level, solve this level via multigrid, etc. This then leads to the full multigrid cycle shown in Figure 4.8, where we restrict attention to a V-cycle algorithm, although it should be understood that the general μ cycle could easily be incorporated. The V-cycle algorithm is the same as before.

4.7 The Full Approximation Storage Scheme (FAS)

We again start from our general boundary value problem

$$Lu = f \tag{4.26}$$

where L may now be a non-linear elliptic operator. This is again differenced to

$$L^h u^h = f^h.$$

Now the local truncation error, τ^h is defined by:

$$\tau^h \equiv L^h u - f^h. \tag{4.27}$$

where u is the continuum solution. Note that if we could somehow compute τ^h then we see that we could solve

$$L^h u^{*h} = f^h + \tau^h \tag{4.28}$$

and the solution, u^{*h}, of this equation would be precisely u. Now if an approximation of τ^h could be determined then we could get a better solution to the difference equations. Later, we will see how this viewpoint will aid us in interpreting the FAS scheme.

We now derive the FAS equations. Recall that we are solving

$$L^h u^h = f^h.$$

with L^h assumed to be non-linear. We again define a residual, r^h:

$$r^h \equiv L^h \tilde{u}^h - f^h. \tag{4.29}$$

Now we can still think about the problem as trying to compute a smooth correction v^h such that

$$u^h = \tilde{u}^h + v^h \tag{4.30}$$

but we will need to get a modified form for the coarse grid equation. If we let u^h in the above correction be just a new value of $\tilde{u}^h$ then this equation becomes

$$\tilde{u}^h := \tilde{u}^h + v^h$$

which is actually

$$\tilde{u}^h := \tilde{u}^h + I_H^h v^H \tag{4.31}$$

since the correction will be determined from the coarse grid.

We then consider (as mentioned previously)

$$\begin{aligned} L^h\left(\tilde{u}^h + v^h\right) - L^h \tilde{u}^h &= L^h u^h - L^h \tilde{u}^h \\ f - L^h \tilde{u}^h = -v^h & \end{aligned} \tag{4.32}$$

Provided $\tilde{u}^h$ is smooth, we can then formulate a coarse grid version of this relation:

$$L^H u^H - L^H I_h^H \tilde{u}^h = -I_h^H r^h \tag{4.33}$$

or

$$L^H u^H = L^H I_h^H \tilde{u}^h - I_h^H r^h. \tag{4.34}$$

This is the FAS coarse grid correction equation. Once an appropriately accurate solution $\tilde{u}^H$ has been computed to the above equation we update $\tilde{u}^h$ as follows

$$\tilde{u}^h := \tilde{u}^h + I_H^h \left(u^H - I_h^H u^h\right). \tag{4.35}$$

Brandt points out[9] that the more obvious update, $u^h := I_H^h \tilde{u}^H$, is inferior since it would introduce the interpolation errors of $\tilde{u}^H$ instead of the interpolation errors of only the correction v^H. (In other words, equation (4.35) preserves smoothness information encoded in u^H prior to the CGC, whereas the other update throws such smoothness information away)

To see one of the major advantages of using multigrid rather than other iterative techniques such as those discussed in Chapter 3 , and also to get a different perspective on the FAS equations, we shall take another look at equation (4.22). We recall that the residual vector is defined as

$$r^h = L^h \tilde{u}^h - f^h.$$

Thus, if we rewrite equation (4.22) using this definition of the residual vector we get

$$\begin{aligned} L^H u^H &= L^H I_h^H \tilde{u}^h - I_h^H \left(L^h \tilde{u}^h - f^h\right) \\ &= \tau_h^H + f^H \end{aligned} \tag{4.36}$$

where

$$\tau_h^H \equiv L^H I_h^H \tilde{u}^h - I_h^H L^h \tilde{u}^h. \tag{4.37}$$

Recall that when we discussed the FAS scheme, we introduced the notion of correcting the equation with the local truncation error, and discussed how this would make the difference solution a better approximation of the continuum solution. Similarly, we can interpret τ_h^H as an estimate of the local truncation error of the level-H equations relative the level-h ones. This leads to what Brandt has termed the dual point of view since we can now view the fine grid as a device for calculating a correction τ_h^H to the coarse grid equations. Previously we only viewed the coarse grid as a device for accelerating convergence on the fine grid. We now give the algorithm for the full multigrid code using the FAS scheme in Figure 4.11.

The τ that is generated in the FAS scheme is an estimate of the relative local truncation error, τ_h^H. We see this by looking at the local truncation error, τ^H [9] defined by

$$\tau^H \equiv L^H\left(\tilde{I}^H u\right) - I^H\left(Lu\right) \tag{4.38}$$

where I^H and $\tilde{I}^H$ are two continuum to H transfer operators. We now see a direct analogy with equation (4.37). As $\tilde{u}^h \rightarrow u$ then $\tau_h^H \rightarrow \tau^H$. Because of this analogy τ_h^H is also referred to as the relative local truncation error. In traditional multigrid applications local truncation error estimates are often used to provide natural stopping criterion for the overall iterative process, *i.e.* once the residual becomes appreciably less than τ_h^H we can deem the problem solved. We will see later that τ_h^H can also be used in grid adaptation criteria.

4.8 Richardson extrapolation

Richardson[44] asserted a long time ago that if a grid function $\tilde{u}$ satisfies an $O\left(h^2\right)$ centered difference approximation to some PDE, then, in the limit $h \rightarrow 0$, we can expect the continuum solution, u, to have an expansion of the form

$$u = \tilde{u} + h^2 e_2 + h^4 e_4 + h^6 e_6 + \cdots, \tag{4.39}$$

where $e_2, e_4, \cdots$ are error functions which are independent of the mesh spacing h. An extremely good explanation of Richardson extrapolation can be found in Choptuik[11] . Our application of Richardson extrapolation is quite simple. [2] Our basic approach is to solve our discrete system of equations at two resolutions, $2h$ and h, and then assume

$$\begin{aligned} u^h &= u + h^2 e_2 + O\left(h^4\right) \\ u^{2h} &= u + (2h)^2 e_2 + O\left(h^4\right) \end{aligned}$$

to obtain the fourth-order accurate approximation

$$\frac{4}{3}u^h - \frac{1}{3}u^{2h} = u + O\left(h^4\right) \tag{4.40}$$

Thus, once the second order solutions have been determined we can cheaply compute a fourth-order accurate solution.

As should be expected, our experience shows us that Richardson extrapolation only works well when the errors, $u - u^h$ are small. A problem with our straightforward application of Richardson's ideas is that it depends rather crucially on the uniform application of centered differences. For non centered difference stencils, such as those we typically use on

[2] R̈uede [45] has come up with much more general versions, however we will not encounter these complications.

the boundaries, the error can not be assumed to admit only even terms. Thus, extra care must be taken when treating boundaries in order to ensure that our grid functions will have expansions (shown in Chapter 6) .

One could also theorize that by using three levels, we could obtain $O\left(h^6\right)$ accuracy, but this of course assumes that the leading order error term is now up to $O\left(h^4\right)$. In our applications we generally find that the leading order error term is only smooth up to $O\left(h^2\right)$ which makes it difficult to obtain accuracy greater than $O\left(h^4\right)$.

In multigrid solvers, we pointed out that one can use the local relative truncation error estimate to obtain stopping criteria. In order to use Richardson extrapolation we must drive the residuals low enough to ensure that we are computing smooth error terms properly.

Richardson extrapolation can not be used for general solutions to PDE's since in general these solutions may not be smooth. Since we expect our equation (the Hamiltonian constraint for black hole spacetimes) to admit only smooth solutions, we can expect to be able to use this powerful technique [18][12][20][13] .

4.9 Deferred correction

We have seen the complications of finite differencing boundary conditions when we adopt coordinates which are not boundary conforming. We then pointed out that a way around this problem is to use deferred correction. This idea was originally suggested by L. Fox and then Brandt[9].

We can understand the basic idea behind deferred correction by recalling the discussion of Section 4.7, where we observed that we could improve the accuracy of a finite difference solution by adding the truncation error to the right-hand-side of the difference equation. In the FAS scheme, we saw that we could use the relative truncation error estimate to modify the right hand side of the coarse grid equations to get a better estimate of the solution on a finer grid. The methodology here is similar. We will modify the right hand side of the difference equations on the finest grid to get a higher order approximation of the original system of equations which the difference equations were approximating.

Through experience (ours and others) we know that a good general strategy when finite-differencing is to first use low order techniques since they are much easier to develop, they are typically more efficient[9] and they can often be used as components in a subsequently developed higher-order program.

Consider again the general problem

$$Lu = f$$

We let

$$L_p u = f \tag{4.41}$$

be an $O(h^p)$ approximation of the continuum problem, so that

$$L_p u = f + O(h^p)$$

and let

$$L_q u = f \tag{4.42}$$

be a distinct and lower order finite-difference approximation, so that

$$L_q u = f + O(h^q) \qquad q < p.$$

Given these two distinct difference systems, the idea of deferred correction is to use an iterative process defined by:

$$L_q u^{(n+1)} = f + L_q u^{(n)} - L_p u^{(n)}. \tag{4.43}$$

Now it is clear that if this iteration converges, *i.e.* if

$$u^{(n+1)} \rightarrow u^{(n)},$$

then $L_q u^{(n+1)}$ cancels out $L_q u^{(n)}$ in equation (4.43). Thus, equation (4.43) becomes

$$L_p u^{(n)} = f \tag{4.44}$$

and we see that the grid function does indeed satisfy the higher order equation.

Brandt[9] points out two interesting comments concerning deferred correction. The first comment is that the total work will be proportional to the higher order operator and not the lower order operator. Thus, the hope of having a faster solver by using lower order techniques is essentially gone, unless very few points in the domain use deferred correction. We only use deferred correction for the boundary points, which would signify that the overall work will still be proportional to the lower order scheme. Since we wish to use Richardson extrapolation, we must drive the residuals down very low, which basically means that our overall rate of convergence will be consistent with the higher order operator. The second comment is that the higher order stencil need not be stable. This is due to the fact that the convergence is fast only in smooth components, for which the lower order operator is a good approximation to the higher order operator. The convergence is slow only for high-frequency components. Since the instability is only a property of high-frequency components it will usually come on slowly, which means that we may have already driven the residuals down low enough to determine a solution.

Thus, our scheme is set to handle general boundaries. There is no problem discretizing the interior equations to second order,

$$L_2^h \tilde{u}^h = f^h. \tag{4.45}$$

The boundary equations are much more problematic since the coordinate system is not boundary conforming, so we will use first order differencing for the boundaries, namely,

$$B_1^h \tilde{u}^h = g^h. \tag{4.46}$$

We then use deferred correction only on the finest grids[3]right before the coarse grid correction. Thus, we iteratively solve the system

$$\begin{aligned} L_2^h \tilde{u}^h &= f^h \\ B_1^h \tilde{u}^h &:= g^h + \left(B_1^h \tilde{u}^h - B_4^h \tilde{u}^h\right). \end{aligned} \tag{4.47}$$

We use fourth order deferred correction since our hope is to use Richardson extrapolation to fourth order. To understand why we need to use an equation which is greater than second order, we look at an example. The one dimensional version of the inner boundary condition (2.31), is

$$\frac{\partial u}{\partial r} = -\frac{u}{2a} \tag{4.48}$$

and we can finite difference this to second order, using

$$\begin{aligned} u\,(a+\xi) &= u + \xi u_r + \tfrac{\xi^2}{2} u_{rr} + \tfrac{\xi^3}{6} e_2 + \tfrac{\xi^4}{24} e_3 \\ u\,(a+h+\xi) &= u + (\xi+h)\, u_r + \tfrac{(\xi+h)^2}{2} u_{rr} + \tfrac{(\xi+h)^3}{6} e_2 + \tfrac{(\xi+h)^4}{24} e_3 \\ u\,(a+2h+\xi) &= u + (\xi+2h)\, u_r + \tfrac{(\xi+2h)^2}{2} u_{rr} + \tfrac{(\xi+2h)^3}{6} e_2 + \tfrac{(\xi+2h)^4}{24} e_3 \end{aligned}$$

where ξ is shown in Figure 3.6. When we solve these equations for u_r we obtain

$$\begin{aligned} u_r &= \alpha u\,(a+\xi) + \beta u\,(a+h+\xi) + \gamma u\,(a+2h+\xi) \\ &+ e_2 h^2 + e_3 h^3 \end{aligned} \tag{4.49}$$

where α,β, and γ are constants, and e_2 and e_3 are the coefficients of the error terms. We see that the error contains terms in h^3. In order to eliminate the error term in front of h^3 we must use at least third order expansions. Empirically we have found that by using only third order differencing, the error term is not smooth, and thus we can not extrapolate. Therefore, we use fourth order differencing to eliminate the third order error term, and to have a smooth fourth order error term.

[3]When one uses a FMG cycle, then we use deferred correction on all of the finest grid used in each cycle.

4.10 Adaptive Mesh Refinements (AMR)

Now that we have discussed methods to boost the accuracy of the solution, we are ready to discuss AMR. Non-uniform resolution is needed in most practical problems. For example, increasingly finer grids are needed near singularities, and near non-smooth boundaries. We use increasingly coarser grids since the initial value problem is defined on an unbounded domain. We found that some useful discussions on AMR are found in Choptuik [15],Berger & Oliger[2] and Brandt[8]. We use the AMR techniques which follow directly from Choptuik's work which followed from Brandt's[8] work, and from Berger & Oliger's work.

We recall from Section 4.1 that the algorithms use a series of discretization levels $l = 0, \ldots, l_{\max}$ where $l = 0$ is the coarsest level, and $l_{\max}$ is the finest and $h_l = 2h_{l+1}$ Now the basic approach to AMR is the following.

Using an FAS scheme, we compute τ_h^H on all the points on the coarse grid which have corresponding fine grid points. We then specify a maximum τ, $\tau_{\max}$. We refine all the points that have $\tau_h^H > \tau_{\max}$. Thus, we instantly come up with a scheme for adaptive gridding.

4.11 Overview of the Berger & Oliger approach to mesh refinement

There are four major stages in the Berger & Oliger regridding algorithm:

(1) flag points needing refinement,
(2) cluster the flagged points,
(3) generate a grid for each cluster,
(4) evaluate the quality of the regridding and possibly repeat the steps.

We can incorporate the entire regridding process in the first cycle of a FAS scheme. This is the case since all we need is an *estimate* for the relative truncation error: the values of τ_h^H generated just before the first coarse grid correction is initiated are generally acceptable.

4.11.1 Flag points needing refinement

An attractive feature of a FAS scheme is that it naturally generates a relative truncation error estimate. Given a maximum allowable absolute value, $\tau_{\max}$, for τ_h^H we define a characteristic function, `char` , of 1's and 0's where

$$\begin{aligned} \texttt{char} = 0, &\quad \tau_h^H \leq \tau_{\max} \\ \texttt{char} = 1, &\quad \tau_h^H > \tau_{\max}. \end{aligned} \tag{4.50}$$

This characteristic function is generated whenever we need to regrid a level.

Note that the concept of "grid" is now distinct from that of "level". Previously we could assume that each level contained one grid, but now there is a possibility of multiple grids on any given level (except perhaps for the coarsest). For example, Figure 4.12 shows a one dimensional grid with two refinements.

4.11.2 Cluster the flagged points

The most difficult part of any regridding algorithm is the clustering algorithm. On one hand, if the clustering algorithm does a bad job, meaning that it clusters together more non flagged points than flagged points, then there will be a lot of work wasted at the regions that do not have to be refined. On the other hand, if the clustering algorithm does an extremely good job in clustering the points, but takes an enormous amount of computational time, then this too becomes wasteful. We derived three tables (4.1,4.2,4.3) to show the efficiency of the refinement and as an aid in designing a clustering algorithm.

The % usefulness is the ratio of the total number of points minus the boundary points divided by the total number of points. In the Berger and Oliger approach, values at boundary points of refined grids are typically defined via interpolation in parental meshes and hence can not contribute to an increase in accuracy. Choptuik points out that when the % usefulness falls below 50% then the efficacy of the refinement is questionable. Thus, for the 3 dimensional case, we will need $n \approx 17$ before we should regrid. Note, however, that we are not demanding that we must always use cubes in three dimensions, *i.e.* we can use grids which contain $n_x \times n_y \times n_z$ points. Our first demand is that these dimensions be at least 9 points, $(n_x > 9, n_y > 9, n_z > 9)$.

Multigrid algorithms work with an odd number of points since each level discretization is based on a factor of 2 of the other levels, *i.e.* if a fine grid had an even number of points, n, then the number of points on the coarse grid would be

$$N = 2n - 1,$$

where N is the number of points on the fine grid, hence N could not be an integer. The clustering algorithm demands that each dimension must contain an odd number of points. If we have flagged only an even number of points we then add an additional point.

Since in one dimension a grid is just an interval, clustering is trivial. An appropriate algorithm is given in Figure 4.13. We see that this algorithm can have clusters/grids which overlap each other. When this occurs, the clusters should be merged otherwise boundary conditions will not be uniquely defined and/or there will be unnecessary inter-grid communications on the same level.

Table 4.1: 1D uniform point refinement efficiency

n	n_{total}	$n_{boundary}$	% useful
2	2	2	0.0
3	3	2	33.3
5	5	2	60.0
9	9	2	77.8
17	17	2	88.2
33	33	2	93.9
65	65	2	97.0
129	129	2	98.4

Berger & Oliger[2] point out that a good clustering algorithm should serve two purposes. The first is to separate spatially distinct phenomena so that different features will be in separate grids. The second purpose is to subdivide the points on the grid, when the efficiency becomes low. This efficiency is of course different from the efficiency we talked about previously. We calculate this efficiency as the ratio of the total number of points in the cluster with `char` equal to one by the total number of points in the cluster. This estimates the fraction of the area of the rectangle that needed to be refined.

For now on we shall call this the rectangle efficiency. Unlike Berger & Oliger we abandon rotated rectangular domains.

Figure 4.15 shows an example of the grid refinement process. Here the filled in points represent those points which have `char`=1. The first step is to draw a rectangle around the flagged points. This is shown in Figure 4.16. In step two the efficiency of these two grids is calculated to be 52% and 43%. Step three then sees which grids have an efficiency less then `eff_small` and determines that grid two must be subdivided.

After the `char` is calculated in this region, the new rectangles are then drawn. Figure 4.17 displays the new rectangles. We see that some originally flagged points are then not inside the new rectangles. The final step is to draw rectangles around these missing points, which is shown in Figure 4.18.

The clustering algorithms are shown in Figure 4.14.

Figure 4.8: A general FMG algorithm.

```
PROCEDURE Full_Multigrid_CS_SCHEME( l_max, ncyc, pre, pst )
   Initialize Variables
   Initialize Memory
   SOLVE(L^H u^H = f^H)     Solve the coarsest level problem
   DOl = 1, l_max
      DOcycle= 1 , ncyc
      u^h := II_H^h u^H)
      VCycle( cycle, l , pre, pst)
      END DO  for cycle
   END DO  for l
END PROCEDURE
```

Table 4.2: 2D uniform square refinement efficiency

n	n_{total}	$n_{boundary}$	% useful
2	4	4	0.0
3	9	8	11.1
5	25	16	36.0
9	81	32	60.5
17	289	64	77.9
33	1089	128	88.2
65	4225	256	93.9
129	16641	512	96.9

Table 4.3: 3D uniform cubical refinement efficiency

n	n_{total}	$n_{boundary}$	% useful
2	8	8	0.0
3	27	26	3.7
5	125	98	21.6
9	729	386	47.1
17	4913	1538	68.7
33	35937	6146	82.9
65	274625	24578	91.1
129	2146689	98306	95.4

Figure 4.9: A schematic of the V-cycle used in MG algorithms.

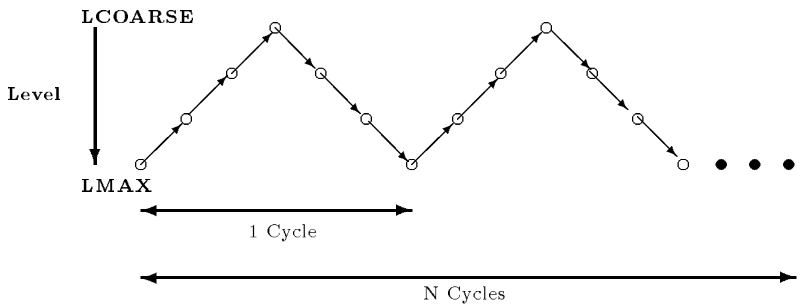


Figure 4.10: A schematic of the W-cycle used in MG algorithms.

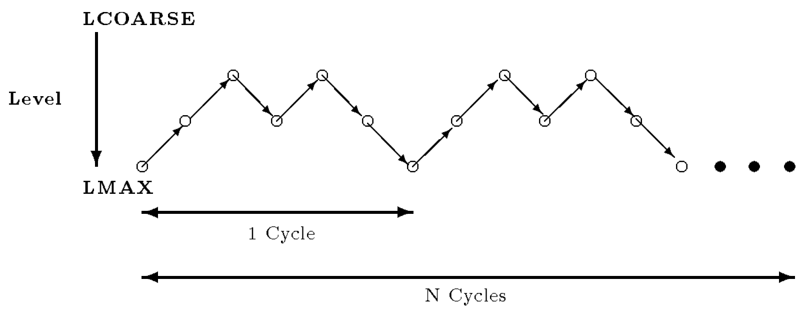


Figure 4.11: The full multigrid FAS algorithm.

PROCEDURE FullMultigrid_FAS($l_{\max}$, ncyc, pre, pst)
 Initialize Variables
 Initialize Memory
 SOLVE($L^H u^H = f^H$) *Solve the coarsest level problem*
 DOl $= 1, l_{\max}$
  DOcycle= 1 , ncyc
  $u^h := \Pi_H^h u^H$
  VCycle_FAS(cycle, l , pre, pst)
  END DO *for* cycle
 END DO *for l*
END PROCEDURE

PROCEDURE VCYCLE_FAS(cycle, $l_{\max}$, pre, pst)
 Cycle up to the coarsest level
 DOl $= l_{\max}, l_{\text{coarse}} + 1$
   DOp = 1 , pre
    RELAX *Smooth the residuals*
   END DO
  END IF
   $\tau_h^H = L^H I_h^H u^h - I_h^H L^h u^h$ *determine the truncation error estimate*
   $f^H = I_h^H f^h + \tau_h^H$ *Correct the rhs*
 END DO

 Now solve the system on the coarsest level
 SOLVE($L^H u^H = f^H$)
 Now come back down to the finest level, performing the needed corrections
 DOl $= l_{\text{coarse}}, l_{\max} - 1$
  Determine the correction
  $u^h := u^h + I_H^h \left(u^H - I_h^H u^h\right)$
  Now smooth out u^h
  DOq = 1 , pst
   RELAX *Smooth the residual*
  END DO
 END DO
END PROCEDURE

Figure 4.12: A schematic view of two levels, where the finer level contains two grids.

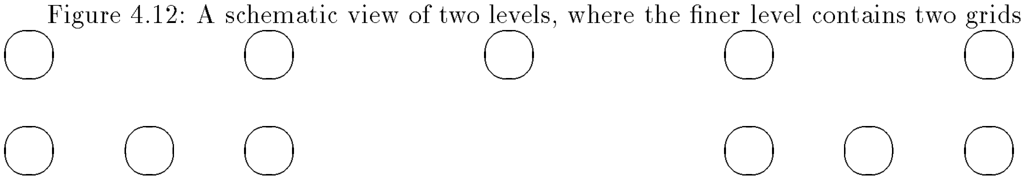


Figure 4.13: A basic, 1D clustering routine

```
PROCEDURE Cluster_1D( char,gmin,gmax )
   CLUSTER= 0 ,   So far, zero clusters have been defined
   DOj = 1 , n    Loop on all the points on the domain j=1 ··· n
      IF( char(j)=1 and char(j-1)=0) THEN
         CLUSTER:=CLUSTER+ 1    We found a new cluster
         gmin( CLUSTER) = j
      END IF
      IFchar(j)=1 and char(j+1)=0 THEN
         gmax(CLUSTER) = j    This is the end of the cluster
      END IF
   END DO
   Now we check to see if there if an odd number of points defined on each grid
   DOj = 1 , CLUSTER
      IF mod ( 1 + gmax(j) - gmin(j) ) ,2 ) =0 THEN
      If the grid contains an odd number of points then
          gmax(CLUSTER) := gmax(CLUSTER) + 1
         Add extra point in arbitrary direction
      END IF
   END DO
END PROCEDURE
```

Figure 4.14: A basic multidimensional clustering routine

```
PROCEDURE Cluster_nd( char,gmin,gmax,τ_max,τ_h^H,δ,grids )
   DRAW_RECTANGLES( char,gmin,gmax,grids )
   DOi = 1, grids   Loop on all the grids now
      ε = CALCULATE_GRID_EFFICIENCY( char,gmin(i),gmax(i) )
      IF (ε  <  ε_min ) THEN   ε_min is in general 50%
         τ_max := τ_max − δ
         Get_Char( char, τ_h^H , τ_max )
         Cluster_nd( char,gmin,gmax,τ_max,τ_h^H,δ )
      END IF
   END DO
END PROCEDURE
```

Figure 4.15: The flagged points in an AMR routine.

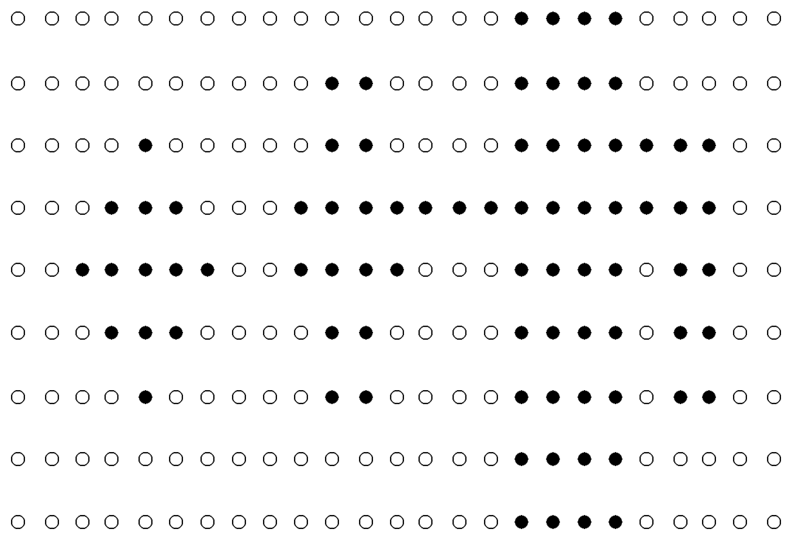


Figure 4.16: The rectangles generated from the flagged points in an AMR routine.

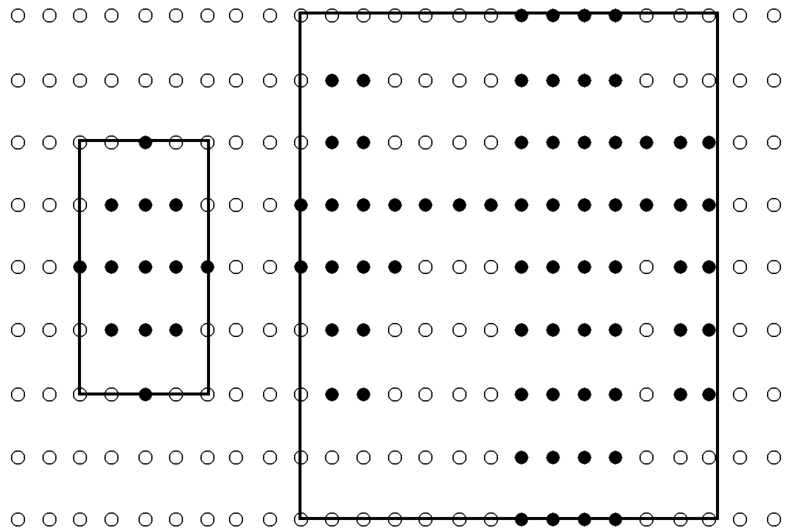


Figure 4.17: The revised rectangles generated from the flagged points in an AMR routine.

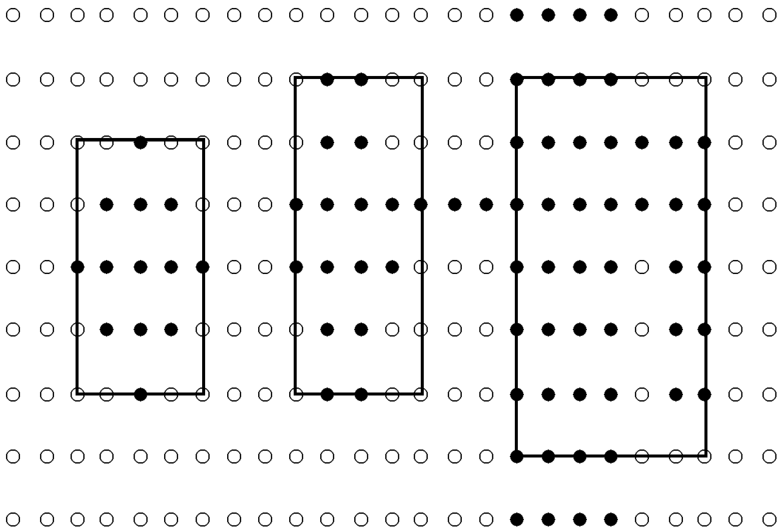
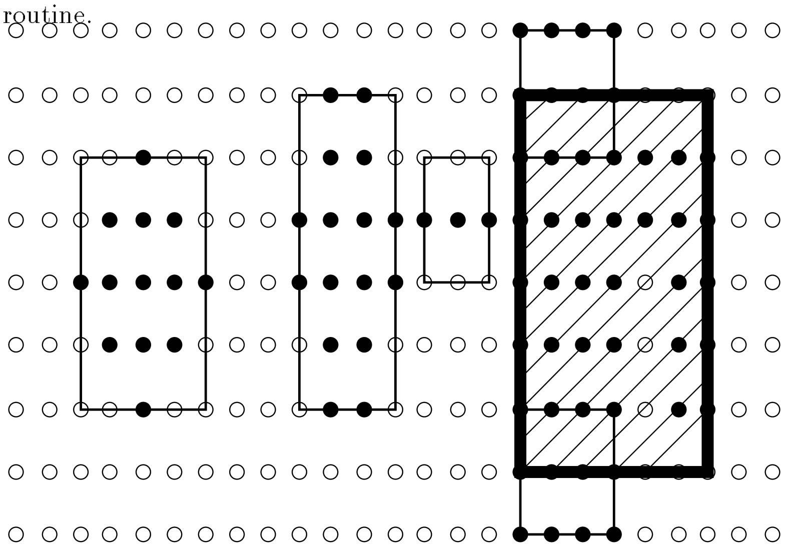


Figure 4.18: The final revision of the rectangles that were generated from the flagged points in an AMR routine.



4.11.3 Grid generation

The grid generation algorithm takes `gmin`, an array which contains the smallest value of x,y and z, `gmax` , the largest value of x,y and z, and `grids` , the number of independent cluster regions, and then determines make sure that each dimension of the rectangular region contains an odd number of points and that each dimension is also greater than or equal to the minimum allowable number. If any of these criteria are not satisfied `gmin` and `gmax` must be modified as to satisfy them.

The grid generation routine defines all the grid functions on the new grid, sets up the links to the tree structures[16] and then initializes the functions on this grid.

4.11.4 Memory management

A crucial task in designing a MLAT algorithm is designing the data structures. Here we utilize Choptuik's[16] tree manipulations and data structures.

All the memory for the grid functions is allocating in a single, one-dimensional array `q`. We typically manipulate grid functions on two grids simultaneously: for example we need a coarse and a fine grid in order to define a truncation error estimate. Thus, we include separate pointers for coarse and fine grid functions. We use pointers which are shown in Figure 4.19. `qptr_type1(g)` is a vector which contains an integer value pointing to the beginning of the the storage for grid `g` of size `type1`. We now introduce the possibility of various grid types which we label as `qptr_type1, qptr_type2` .... Thus, `qptr_type2(g)` would then contain the integer value of where the storage of grid `g` begins in `q`. In general we can have an arbitrary number of types. Each type has an associated number of values for each `g`. The number of values associated with each `g` is then determined by the length of each array of the type multiplied by the number of functions which are of this length, `nfcn_type1`. The functions are given unique names which are then stored in a common block . These common blocks are then `EQUIVALENCE`d to another pair of pointers which are of length `nfcn_typen`, where `typen` refers to the set of all types. This pair is set to distinguish between the coarse and fine points. The storage scheme is shown in Figure 4.19. Storage can then be allocated and deallocated dynamically since the only change in memory is in the values of the `q` pointers.

Whenever we need to address grid functions for a particular set of grids, we first call a routine which loads up the coarse and fine grid functions for the particular two grids we wish to address. We then reference these arrays such as `q(uc)` which would then point to the place in memory where the u function is defined for the coarse grid.

Figure 4.19: Memory management.

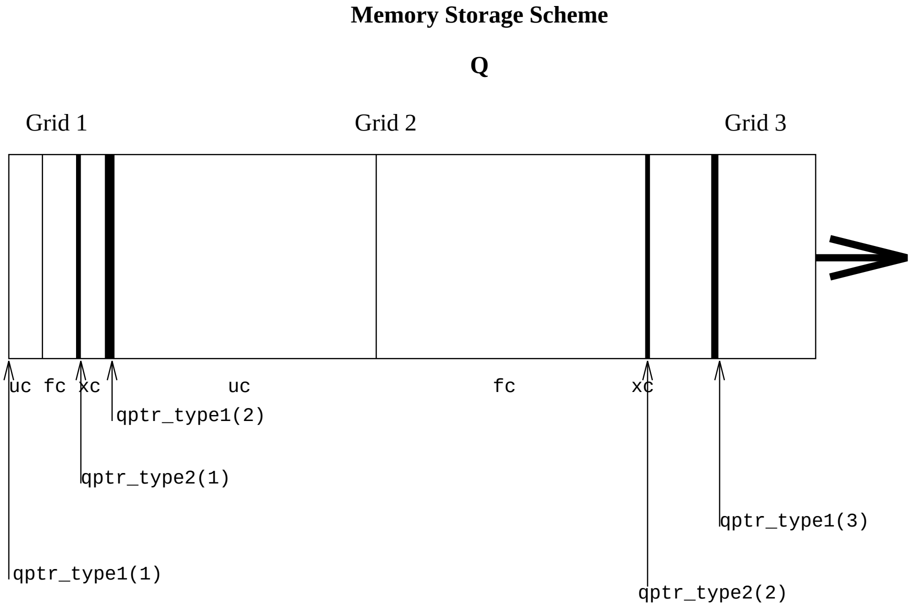


Table 4.4: Grid Links for tree structure

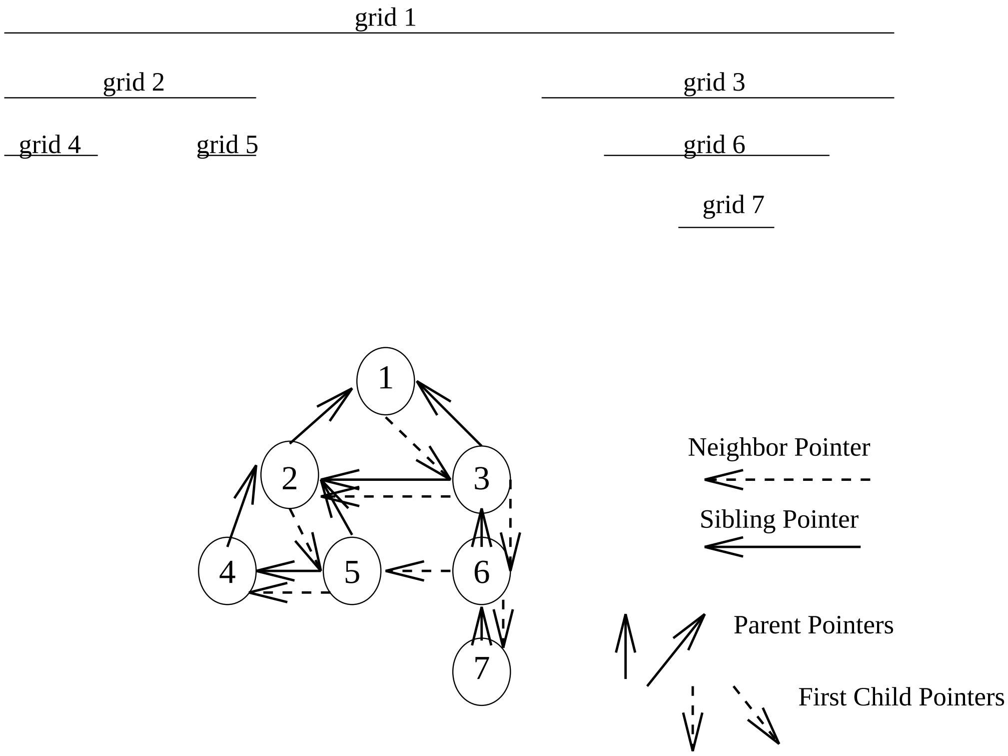
g	gpp	gpch	gpsib	gpngh
1	UNDEF	3	UNDEF	UNDEF
2	1	5	UNDEF	UNDEF
3	1	6	2	2
4	2	UNDEF	UNDEF	UNDEF
5	2	UNDEF	4	4
6	3	7	UNDEF	5
7	6	UNDEF	UNDEF	UNDEF

4.11.5 Tree structures

Tree structures are the natural data structure for AMR using the Berger and Oliger[2] approach. We use Choptuik's `Fortran 77` implementation of the tree structure[16]. This approach uses the linked lists[32] `lhead`, `gpp`, `gpngh`, `gpsib`, `gpch`. A sample linked-list structure is shown in Figure 4.20. `lhead` is a list of grids which are the last grids of each level. Thus, `lhead`(l_{coarse}) is the head grid of the coarsest level and `lhead`($l_{\max}$) is the head grid of the finest level. `gpp`(`g`) is the parent of `g`, whereas `gpch`(`g`) is the first child of `g`. Likewise `gpsib`(`g`) is the sibling of `g` and `gpngh`(`g`) is the neighbor of `g`. The difference between a sibling and a neighbor, is that siblings have the same parent, and neighbors do not necessary have the same parent.

To better understand this we will give an example. Figure 4.20 shows a one dimen-

Figure 4.20: Adaptive grids shown with tree pointers.



sional grid with various refinement regions. Here there are 4 levels. The tree structure is also shown in Figure 4.20. Here `UNDEF` is a grid which has not been defined. Now to traverse all the grids at a given level, all we would have to do is determine the head of the level, via $\mathtt{g} = \mathtt{lhead}(l)$, and then each grid on the level can be scanned by $\mathtt{g} = \mathtt{gpngh}(\mathtt{g})$. One can then chain through the list using the algorithms shown in Figure 4.21.

We have now defined all of the major components in our MLAT codes. We discussed the multigrid Linear Correction Scheme and FAS scheme, the AMR schemes, deferred correction, and Richardson extrapolation. We will examine these schemes in Chapter 6 to demonstrate how we will be able to develop an extrapolatable MLAT code to model the initial value problem in Cartesian coordinates.

Figure 4.21: Algorithms to chain through the tree structure.

```
PROCEDURE CHAIN_ALL_LEVELS( )
   This routine will chain through all the defined grids, starting
   with the coarsest level. This routine will print out the grid number.
   l = l_coarse
   grid = lhead(l)
   START:
      IF( grid = UNDEF ) GOTO END1
      This is the last list defined at the level
      PRINT grid
      grid = GPNGH(grid)
      GOTO START
   END1
   grid = lhead(GL(l + 1)
   The grid number is now the starting grid of the next finer level
   IF ( grid = UNDEF ) GOTO END2
   GOTO START
   END2
END PROCEDURE

PROCEDURE PRINT_SIBLINGS(grid)
   Given a grid, this routine will print out its siblings
      PRINT "THE SIBLINGS OF THIS GRID ARE:"
   START
      grid = GPSIB(grid)
      IF ( grid = UNDEF ) GOTO END
      PRINT grid
      GOTO START
   END
END PROCEDURE
```

Chapter 5

`genpsi`: A three dimensional code for the solution of the Hamiltonian Constraint

Most of the contents of this chapter have been published in the paper, "Three-dimensional initial data for the collision of two-black holes"[20]. The only change that we make in this dissertation is to point out at least one correction to the original code. In the original version, there were a number of mistakes in the stencils which were defined for the sweep in the z direction. To alleviate this problem, we removed the sweep in the z direction. In this code, we made no use of the more powerful techniques discussed in the last chapter. Our original motivation for developing a code which used the techniques of the previous chapter derived from flaws of the code described in this chapter. This code also served as a test-bed, where we were able to determine if a Cartesian code could generate accurate initial data.

5.1 The basic approach of `genpsi`

This approach to the numerical solution of the Hamiltonian constraint, equation (2.29), for a pair of black holes employs a finite-difference method based on the usual Cartesian coordinates (x, y, z). The algebraic equations which result from finite-differencing in these coordinates are solved by a variant of line-SOR (Successive OverRelaxation) [50].

There are two clear, major disadvantages with the Cartesian approach we outline in this section. The first is that the coordinates do *not* conform to the inner boundaries ("holes") of the problem domain. This means that the formulation of accurate differenced versions of the boundary conditions is considerably more involved here than in the other two approaches described in this paper. Secondly, because we use a *uniform* Cartesian grid (*constant* mesh increment, h, in each of the coordinate directions), the combination of 1) the need to resolve steep gradients near the holes and 2) limitations on our computational resources, places a severe restriction on the radius at which the outer edge of the computational domain is located. In practice this means that we must impose the asymptotic condition (2.34) in a regime where the neglected terms in the multipole expansion are still significant. These shortcomings became apparent in the numerical results discussed in the Results section of [20].

In Cartesian coordinates, equations (2.29), (2.31) and (2.34), are

$$\begin{aligned} \frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}+\frac{\partial^2\psi}{\partial z^2} &= -\frac{1}{8}\psi^{-7}\bar{A}^2, && (5.1)\\ n_\alpha^x\frac{\partial\psi}{\partial x}+n_\alpha^y\frac{\partial\psi}{\partial y}+n_\alpha^z\frac{\partial\psi}{\partial z} &= -\frac{\psi}{2r_\alpha}\bigg|_{B_\alpha}, && (5.2)\\ x\frac{\partial\psi}{\partial x}+y\frac{\partial\psi}{\partial y}+z\frac{\partial\psi}{\partial z} &= \frac{1-\psi}{r}. && (5.3) \end{aligned}$$

where

$$\begin{aligned} \bar{A}^2 &\equiv \bar{A}_{ij}\bar{A}^{ij}, && (5.4)\\ r &= \sqrt{x^2+y^2+z^2}, && (5.5)\\ r_\alpha &= \sqrt{(x-x_\alpha)^2+(y-y_\alpha)^2+(z-z_\alpha)^2}, \qquad \alpha=1\,,\,2, && (5.6) \end{aligned}$$

$\vec{n}_\alpha=(n_\alpha^x,n_\alpha^y,n_\alpha^z)$ are the unit normals to the holes, and the "centers" of the holes are located at $\vec{C}_\alpha=(x_\alpha,y_\alpha,z_\alpha)$. Equation (5.2) holds at the hole surfaces and equation (5.3) is imposed at the outer boundary. The non-linear Hamiltonian constraint is solved iteratively using a linearization which, given an approximate solution ψ_0, determines the new iterate, ψ:

$$\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}+\frac{\partial^2\psi}{\partial z^2}-\frac{7}{8}\bar{A}^2\psi_0^{-8}\;\psi=-\bar{A}^2\psi_0^{-7}. \qquad (5.7)$$

Figure 5.1 schematically illustrates a typical computational domain which might be used for the case of a computation involving a single hole in two dimensions. In addition to the inner boundary points, the computational domain also contains *interior* points (those marked with a + in the figure) and outer boundary points (filled boxes). We further categorize any point on a boundary as a *single-edge*, *double-edge*, or *triple-edge* boundary point according to whether the point has one, two, or three nearest-neighbors, respectively, which lie *outside* of the discrete domain. For example, in Figure 5.1, the four corner points are double-edge boundary points, and the remaining outer boundary points are single-edge. Similarly, there are 6 double-edge *inner* boundary points in the figure, including the point enclosed by the doubled-diamond. As we shall now discuss, this substantial variety in the types of points in the computational domain results in a proliferation of specific difference equations which must be generated in order to produce a reasonably accurate solution.

5.2 Finite differencing

Our finite-difference analogues of the system (5.7), (5.2) and (5.3) are generated using standard $O(h^2)$ centered, forward, and backward approximations to first and second derivatives.

However, as discussed below, we have *not* implemented a scheme with true $O(h^2)$ truncation error; rather we anticipate only $O(h)$ convergence in the limit $h \to 0$ and possibly indeterminate convergence behavior at practical resolutions.

Adopting the usual finite-difference notation $f(ih, jh, kh) \equiv f[i, j, k]$, we employ the following formulae for the x derivatives:

$$\begin{aligned}
\frac{\partial \psi}{\partial x}(ih, jh, kh) &\equiv \frac{\partial \psi}{\partial x}[i, j, k] \\
&= \frac{\psi[i+1, j, k] - \psi[i-1, j, k]}{2h} + O(h^2) \qquad (5.8)\\
&= \frac{-\psi[i+2, j, k] + 4\psi[i+1, j, k] - 3\psi[i, j, k]}{2h} + O(h^2) \qquad (5.9)\\
&= \frac{3\psi[i, j, k] - 4\psi[i-1, j, k] + 3\psi[i-2, j, k]}{2h} + O(h^2), \qquad (5.10)
\end{aligned}$$

$$\begin{aligned}
&\frac{\partial^2 \psi}{\partial x^2}(ih, jh, kh) \equiv \frac{\partial^2 \psi}{\partial x^2}[i, j, k] \\
&= \frac{\psi[i+1, j, k] - 2\psi[i, j, k] + \psi[i-1, j, k]}{h^2} + O\left(h^2\right) \qquad (5.11)\\
&= h^{-2}\Big(-\psi[i+3, j, k] + 4\psi[i+2, j, k] - 5\psi[i+1, j, k] + 2\psi[i, j, k]\Big) \\
&+ O(h^2) \qquad (5.12)\\
&= h^{-2}\Big(2\psi[i, j, k] - 5\psi[i-1, j, k] + 4\psi[i-2, j, k] - \psi[i-3, j, k]\Big) \\
&+ O\left(h^2\right), \qquad (5.13)
\end{aligned}$$

and the obvious counterparts for the y and z derivatives. In order to gain some sense of how this process is carried out for the 3D, two black-hole case, it is instructive to consider the derivation of 2D difference equations associated with the three stencils depicted in Figure 5.1.

We first consider the stencil represented by boxes in Figure 5.1 Using (5.11) and the analogous formula for the y derivative in (5.7) and ignoring any z-dependence, we obtain the difference expression:

$$\begin{aligned}
&\frac{\psi[i+1, j] + \psi[i-1, j] + \psi[i, j+1] + \psi[i, j-1] - 4\psi[i, j]}{h^2} \\
&-\frac{7}{8}\left[\bar{A}^2 \psi_0^{-8}\right][i, j]\psi[i, j] = -\left[\bar{A}^2\, \psi_0^{-7}\right][i, j], \qquad (5.14)
\end{aligned}$$

which may also be applied at any other grid point marked with a +.

Now consider the stencil composed of circles which is centered at the double-circled point. In this case, we use centered-differenced versions of both the interior equation (5.7) and the inner boundary condition (5.2). Thus, in addition to (5.14) we have

$$\begin{aligned}
& n^x[i, j] \quad \left[\frac{\psi[i+1, j] - \psi[i-1, j]}{2h}\right] \qquad (5.15)\\
+ \quad & n^y[i, j] \quad \left[\frac{\psi[i, j+1] - \psi[i, j-1]}{2h}\right] = -\frac{\psi[i, j]}{2r[i, j]}. \qquad (5.16)
\end{aligned}$$

Note that although we have used "second-order" differences to generate this last expression, it is, for a general inner boundary point, only a *first-order* accurate version of the inner boundary condition since the boundary point will, as shown in the figure, generally lie a distance $O(h)$ from the $r = r_\alpha$ surface. Second-order accurate formulae *could* be generated using Taylor series expansion and the governing differential equations, but we have not done so. Now, (5.14) and (5.16) both involve the value $\psi[i, j+1]$, which is defined on a lattice point lying outside the computational domain. Therefore, we solve (5.16) for $\psi[i, j+1]$, and substitute the result in (5.14), yielding an expression involving only the points of the stencil:

$$
\begin{aligned}
h^{-2}\Bigg(& \left[1 - \frac{n^x}{n^y}\right][i,j]\psi[i+1,j] + \left[1 + \frac{n^x}{n^y}\right][i,j]\psi[i-1,j] \\
& +2\psi[i,j-1] - \left[4 + \frac{h}{rn^y}\right][i,j]\psi[i,j]\Bigg) - \frac{7}{8}\left[\bar{A}^2\psi_0^{-8}\right][i,j]\psi[i,j] \\
& = -\left[\bar{A}^2\,\psi_0^{-7}\right][i,j].
\end{aligned}
$$

The stencil marked by diamonds in Figure 5.1 is centered on a double-edge boundary point. In this case, application of centered difference formulae to the interior and boundary equations results in references to *two* points which lie outside of the computational domain. In order to derive a *single* equation for the center point we employ forward differencing in the y direction, and centered differencing in the x direction to get

$$
\begin{aligned}
& \frac{\psi[i+1,j] + \psi[i-1,j] - \psi[i,j+3] + 4\psi[i,j+2] - 5\psi[i,j+1]}{h^2} \\
& -\frac{7}{8}\left(\bar{A}^2\psi_0^{-8}\right)[i,j]\psi[i,j] = -\left[\bar{A}^2\,\psi_0^{-7}\right][i,j]
\end{aligned}
\tag{5.17}
$$

and

$$
\begin{aligned}
n^x[i,j] \quad & \left[\frac{\psi[i+1,j] - \psi[i-1,j]}{2h}\right] + \\
n^y[i,j] \quad & \left[\frac{-\psi[i,j+2] + 4\psi[i,j+1] - 3\psi[i,j]}{2h}\right] = -\frac{\psi[i,j]}{2r[i,j]}.
\end{aligned}
\tag{5.18}
$$

As before, solution of the discrete boundary condition for the value $\psi[i+1, j]$, followed by substitution of the result into the differenced interior equation, yields a single equation involving only the stencil-unknowns:

$$
\begin{aligned}
h^{-2}\Bigg(& 2\psi[i-1,j] - \psi[i,j+3] + \left[4 - \frac{n^y}{n^x}\right][i,j]\psi[i,j+2] \\
& + \left[4\frac{n^y}{n^x} - 5\right][i,j]\psi[i,j+1] - \left[3\frac{n^y}{n^x} - \frac{h}{rn^x}\right][i,j]\psi[i,j]\Bigg) \\
& -\frac{7}{8}\left[\bar{A}^2\psi_0^{-8}\right][i,j]\psi[i,j] = -\left[\bar{A}^2\,\psi_0^{-7}\right][i,j].
\end{aligned}
\tag{5.19}
$$

As a last example of the derivation of our difference equations in Cartesian coordinates, we consider yet again the case of a stencil centered on a boundary point of a hole, but this time for the 3D case. Let the center point have coordinates (ih, jh, kh) and assume that of the point's 6 nearest-neighbors, the three with coordinates $((i-1)\,h, jh, kh)$, $(ih, (j-1)\,h, kh)$ and $(ih, jh, (k-1)h)$ lie outside the numerical domain. Then, by definition, (ih, jh, kh) is a triple-edge boundary point. In such a case, we use forward differences in two of the coordinate directions, and centered differences in the other. Now, as we will discuss shortly, the complete set of difference equations is solved using a line-relaxation method in which *blocks* of unknowns, such as $\psi[i,j,k], \quad i = -n/2 \cdots n/2$ with j and k fixed, are updated ("relaxed") simultaneously. We will refer to the coordinate direction along which a block of unknowns extends as the *scanning* direction. Then, at a triple-edge boundary point, we always forward or backward difference in the scanning direction. To determine the type of differencing to be applied along the other two directions, we examine the components of the normal–the direction having the *smallest* component is the direction in which we use centered differences. In the current example, assume we are scanning in the x direction and that $|n^z[i,j,k]| < |n^y[i,j,k]|$, then we use forward differences in the x and y directions and centered differences in the z direction to get the following discrete forms of the Hamiltonian constraint and inner boundary condition:

$$\begin{aligned}
h^{-2}\Big[&-\psi[i+3,j,k] + 4\psi[i+2,j,k] - 5\psi[i+1,j,k] \\
&-\psi[i,j+3,k] + 4\psi[i,j+2,k] - 5\psi[i,j+1,k] \\
&+\psi[i,j,k+1] + \psi[i,j,k-1] + 2\psi[i,j,k]\Big] \\
&-\frac{7}{8}\left[\bar{A}^2\psi_0^{-8}\right][i,j,k]\psi[i,j,k] = -\left[\bar{A}^2\psi_0^{-7}\right][i,j,k],
\end{aligned} \tag{5.20}$$

$$\begin{aligned}
n^x[i,j,k] \quad & \left[\frac{-\psi[i+2,j,k] + 4\psi[i+1,j,k] - 3\psi[i,j,k]}{2h}\right] + \\
n^y[i,j,k] \quad & \left[\frac{-\psi[i,j+2,k] + 4\psi[i,j+1,k] - 3\psi[i,j,k]}{2h}\right] + \\
n^z[i,j,k] \quad & \left[\frac{\psi[i,j,k+1] - \psi[i,j,k-1]}{2h}\right] = -\frac{\psi[i,j,k]}{2r[i,j,k]}.
\end{aligned} \tag{5.21}$$

Solving (5.21) for $\psi[i,j,k-1]$ and substituting in (5.20) we get

$$\begin{aligned}
h^{-2}\Bigg(&-\psi[i+3,j,k] + \left[4 - \frac{n^x}{n^z}\right][i,j,k]\psi[i+2,j,k] \\
&+ \left[4\frac{n^x}{n^z} - 5\right][i,j,k]\psi[i+1,j,k] - \psi[i,j+3,k] \\
&+ \left[4 - \frac{n^y}{n^z}\right][i,j,k]\psi[i,j+2,k] + \left[4\frac{n^y}{n^z} - 5\right][i,j,k]\psi[i,j+1,k]
\end{aligned}$$

$$
\begin{aligned}
&+2\psi[i,j,k+1] + \left[2 + \frac{h}{rn^z} - 3\frac{n^x}{n^z} + 3\frac{n^y}{n^z}\right][i,j,k]\psi[i,j,k]\Big) \\
&-\frac{7}{8}\left[\bar{A}^2\psi_0^{-8}\right][i,j,k]\psi[i,j,k] = -\left[\bar{A}^2\psi_0^{-7}\right][i,j,k]. \qquad (5.22)
\end{aligned}
$$

Clearly, there are seven other distinct stencils for inner triple-edge boundaries in three-dimensions, all of which can be readily obtained from (5.22) by suitable permutations of the i, j, and k indices.

At this point, the procedure we use to generate our difference equations should be reasonably clear. Altogether, including the various single-, double-, and triple-edge boundary cases, we use 77 distinct stencils in the code.

5.3 Solving the difference equations

The finite-difference discretization of (5.7), (5.2), and (5.3) we have outlined above produces a large, sparse set of linear equations which, as mentioned previously, is solved using an iterative, line-relaxation technique. The "kernel" of the algorithm–a single line-relaxation sweep–can be most clearly defined and understood if we momentarily ignore boundary conditions and focus on the solution of the differenced form of (5.7):

$$
\begin{aligned}
&h^{-2}\Big(\psi[i+1,j,k] + \psi[i-1,j,k] + \psi[i,j+1,k] \\
&\quad +\psi[i,j-1,k] + \psi[i,j,k+1] + \psi[i,j,k-1] - 6\psi[i,j,k]\Big) \\
&\quad -\frac{7}{8}\left[\bar{A}^2\psi_0^{-8}\right][i,j,k]\psi[i,j,k] = -\left[\bar{A}^2\,\psi_0^{-7}\right][i,j,k]. \qquad (5.23)
\end{aligned}
$$

Now, let $\psi^{(m)}[i,j,k]$ denote the values of the stored grid-function after the m-th iteration of the solution process. Then, the equations

$$
\begin{aligned}
&h^{-2}\Big(\psi^{(m+1)}[i+1,j,k] - 2\psi^{(m+1)}[i,j,k] + \psi^{(m+1)}[i-1,j,k]\Big) \\
&\quad -\frac{7}{8}\left[\bar{A}^2\psi^{-8}\right][i,j,k]^{(m)}\psi^{(m+1)}[i,j,k] \\
&\quad = -h^{-2}\Big(\psi^{(m)}[i,j+1,k] + \psi^{(m)}[i,j-1,k] + \psi^{(m)}[i,j,k+1] \\
&\quad +\psi^{(m)}[i,j,k-1] - 4\psi^{(m)}[i,j,k]\Big) - \left[\bar{A}^2\psi^{-7}\right]^{(m)}[i,j,k]
\end{aligned}
$$

define a *line-Gauss-Seidel* (LGS) iteration for the system (5.23). For fixed j and k, (5.24) is a linear tridiagonal system for the $n+1$ unknowns $\psi^{(m+1)}[i,j,k], i = -n/2\cdots n/2$. A complete *relaxation sweep* consists of the solution of $(n+1)^2$ such tridiagonal systems, one for each pairing of j and k–after such a sweep, each unknown has been updated exactly one time. We refer to the iteration defined by (5.24) as x-LGS since the lines of unknowns which are simultaneously updated extend along the x direction. Clearly, we can also define y-LGS and z-LGS iterations.

Gauss-Seidel relaxation generally has a notoriously slow convergence rate, particularly in the limit $h \rightarrow 0$. In order to speed convergence, it is usually helpful to employ the technique of *overrelaxation*. For example, associated with the x-LGS iteration (5.24) is the x-LSOR (Line Successive OverRelaxation) iteration:

$$\psi^{(m+1)}[i,j,k] = \omega \tilde{\psi}^{(m+1)}[i,j,k] + (1-\omega)\,\psi^{(m)}[i,j,k], \tag{5.24}$$

where $\tilde{\psi}^{(m+1)}[i,j,k]$ satisfies the Gauss-Seidel equations (5.24), and ω is the overrelaxation parameter which generally must satisfy $1 \leq \omega < 2$. Although our algorithm incorporates this strategy, with an h-dependent, empirically determined ω, we also use another, somewhat *ad hoc*, technique which basically amounts to an additional overrelaxation applied on a pointwise basis. Again, assuming that we are scanning in the x direction, rather than solving (5.24), we solve

$$\begin{aligned}
& h^{-2}\Big(\psi^{(m+1)}[i+1,j,k] - 4\psi^{(m+1)}[i,j,k] + \psi^{(m+1)}[i-1,j,k]\Big) \\
& -\frac{7}{8}\left[\bar{A}^2\psi^{-8}\right][i,j,k]^{(m)}\psi^{(m+1)}[i,j,k] \\
& = -h^{-2}\Big(\psi^{(m)}[i,j+1,k] + \psi^{(m)}[i,j-1,k] + \psi^{(m)}[i,j,k+1] \\
& +\psi^{(m)}[i,j,k-1] - 2\psi^{(m)}[i,j,k]\Big) \\
& -\left[\bar{A}^2\psi^{-7}\right]^{(m)}[i,j,k].
\end{aligned} \tag{5.25}$$

Together, equations (5.24) and (5.25) define the core of what we call an x-LSOR sweep. Where necessary, boundary equations are simultaneously solved with the interior equations (such as (5.24)), but boundary values are never overrelaxed.

Briefly then, our complete iterative procedure for solving the Hamiltonian constraint proceeds as follows. We perform x-LSOR, and y-LSOR relaxation sweeps [1] in succession until

$$\left\|\psi^{(m+1)}[i,j,k] - \psi^{(m)}[i,j,k]\right\|_1 < \epsilon, \tag{5.26}$$

for some convergence parameter, ϵ, typically 2×10^{-6}. Each d-LSOR sweep ($d = x, y$ or z) requires the solution of $(n+1)^2$ linear systems in $n+1$ unknowns. Each linear system is either 1) *tridiagonal*–a situation which occurs whenever the discrete equations involve only centered differences in the d-direction or 2) *7-diagonal*, if the equations involve forward or backward differences in the d-direction. The tridiagonal systems are solved using a tridiagonal solver which has been optimized for the particular machine architectures (Cray Y/MP, Cray 2) on which the code is run, while the 7-diagonal systems are solved using a bi-conjugate gradient

[1] We do not sweep in the z direction at the time of this dissertation since a bug appears to be in the code for some of the stencils in the z direction.

method. [2] In both cases, the linear system can be solved using $O(n)$ operations, so a complete d-LSOR sweep requires $O(n^3)$ computational work. Empirically, we usually obtain convergence with $O(n)$ sweeps; this represents the optimal asymptotic performance which can be expected for an LSOR method. Thus, our algorithm requires $O(N^{4/3})$ operations to compute a solution on a mesh containing $N \approx n^3$ unknowns. We note that for large N this implies significantly poorer performance than should be possible with a multigrid technique.

5.4 Results and comparison of genpsi to Cook's Čadež multigrid code

5.4.1 Methodology

In this section, we assess and compare the relative performance of **genpsi** to Cook's multigrid algorithm described above by considering the solution of the Hamiltonian constraint for several initial configurations describing two black holes[20]. This comparison is taken taken out of the article co-written by this author [20].

The basic methodology we adopt in our analysis is the straightforward technique of *convergence testing*—for any given physical problem and specific solution technique we generate numerical results at several different *resolutions* (different basic scales of discretization, h, for the finite-difference methods. From these *convergence series* we can then estimate, in the ideal case, 1) the actual level of error, at a given resolution, in any given solution and 2) the rate of convergence of the numerical solution to the continuum solution, again at some specific resolution. This approach was adopted in a previous comparison (Choptuik *et al.*)[13] of different numerical techniques which had been designed for the solution of a particular problem in numerical relativity. The point of this approach is that quantities such as the level of error and convergence rate should be *intrinsically* (i.e. without reference to an analytic solution or a previously computed numerical solution) assessable. However, the error analysis of the basic numerical methods employed in [13] was expedited by the availability of high-accuracy numerical results generated from the application of Richardson-extrapolation techniques to the "raw" output of one of the methods. In the current case as well, the convergence behavior of one of the methods—the Čadež scheme—is such that the output from the algorithm is amenable to Richardson extrapolation. Both theoretical and empirical evidence support our confidence that these extrapolated values are sufficiently accurate to be considered exact for the purpose of assessing the errors and convergence rates of the "raw" output of the three different methods.

[2] For non Cray's we use LINPACK routines for the solution of the tridiagonal systems. The Bi-conjugate gradient method was supplied to us by John Towns[49] at NCSA.

5.4.2 Model Parameters

We have considered 5 different models in our comparison of the methods. As illustrated in Table 5.1, each model is characterized by a considerable number of parameters. Recall from Chapter 2 that α is the dimensionless ratio of the radii of the two holes, while β is the spatial separation of the holes in units of the radius of the first hole, a_1. The three-vectors, $\vec{P}_1$, $\vec{P}_2$, $\vec{S}_1$ and $\vec{S}_2$ can be associated, roughly speaking, with the linear momenta and spins of the two holes—the identification becomes precise only in the limit of large separation of the holes. The parameter $r_{\rm outer}$ differs from the other six in that it has no physical significance—rather, it is the approximate outer radius of the computational domain (the radius at which the Robin boundary condition (2.34) is imposed) used in the corresponding Čadež resolution which, in turn, were used to generate the reference results. Note that in terms of the 6 *physical* parameters there are only three distinct models—A2B8TS (TS since the initial data for this case generates a time-symmetric spacetime), A1B8 and A2B8. Models A1B8NR (NR for NeaR) and A2B8NR differ from their non-NR counterparts only in the setting of $r_{\rm outer}$, which, for the NR computations, was chosen to roughly coincide with the outer "radius" of the computational domain used in the Cartesian calculations. For all of the Cartesian computations, this radius was about $14\,a_1$. By generating reference values with a reduced $r_{\rm outer}$, we are able to assess the effect of the smaller computational domain (relative to the other two algorithms) employed in the Cartesian solutions. The point is that imposing the boundary condition (2.34) at a smaller outer radius, $r'_{\rm outer} < r_{\rm outer}$, does not produce the same solution as that obtained by truncating at $r'_{\rm outer}$ a solution with the boundary condition set at $r_{\rm outer}$.

We ascribe no particular physical significance to the three basic models we consider; the various parameter combinations enumerated in Table 5.1 were chosen to produce relatively interesting and illuminating data sets from the numerical viewpoint, which were also germane to the issue of generating initial data for "realistic" black hole encounters. Thus, based on experience gleaned from axisymmetric computations [19, 23], we chose $\beta = 8$ for the separation parameter in order to produce configurations containing two separate holes (no single, outer apparent horizon or event horizon which envelops both throats), yet where interaction effects are still significant. We include a time-symmetric model providing us with an analytic solution [39], and a useful calibration of our error-assessment. For the models with non-vanishing $\vec{P}_\alpha$, each hole has momentum corresponding to relativistic motion. In addition, the total linear momentum of the "configuration" vanishes, ($\vec{P}_1 + \vec{P}_2 = 0$), so as not to degrade the Robin boundary condition (2.34). The spin vectors, $\vec{S}_\alpha$, endow the holes in the A1B8/A1B8NR and A2B8/A2B8NR models with what we consider "large" and "moderate" amounts of spin, respectively (the A1B8 holes are almost certainly nearly maximal). The cases with non-vanishing $\vec{P}_\alpha$ are "3-dimensional" but, without spin, contain an orbital-plane symmetry. The spin vectors were chosen to explicitly break this remaining symmetry and ensure that the models were truly "generic". Finally, for all models, the coordinate centers of the black holes were located on the z-axis: specifically, for the A1B8 computations, $\vec{C}_1/a_1 = (0, 0, 4)$ and $\vec{C}_2/a_1 = (0, 0, -4)$ while for the A2B8 series, $\vec{C}_1/a_1 = (0, 0, 4.046875)$ and $\vec{C}_2/a_1 = (0, 0, -3.953125)$.

5.4.3 Resolution Parameters

For each method, and for each relevant model listed in Table 5.1, we performed computations at three distinct resolutions—we generically refer to these computations as low, medium, and high resolution runs. For the Cartesian code, these parameters correspond to $h = 64, 96, 128$, and for the Čadež code, we refer the reader to Cook *et al.*[20] to better understand these parameters. For the Čadež scheme, the design of the computational domain, coupled with the fact that meaningful results are attainable on relatively coarse meshes, makes a convergence test using a $4:2:1$ ratio of the low : medium : high resolution scales natural and computationally tractable—thus, in going from a low-to-medium or medium-to-high calculation, all of the Čadež mesh parameters are simply doubled.

As discussed in the previous section, a Cartesian computation is characterized by a single discretization scale, h, or equivalently, by the number of mesh-points, n, on an edge of the computational cube, which, as the resolution is varied, has a fixed physical length. Here, computer resources limit our computations to a maximum $n \approx 128$ (we prefer $n = 16k$, for some integer k from hardware considerations), but solutions generated with $n \approx 32$ are *not* meaningful, so we have produced convergence series using a $2 : 4/3 : 1$ relation of the low : medium : high resolution scales.

5.4.4 Details of the comparison

In our comparison, errors (deviations) are computed at some set of N_{ref} *reference points*, labeled, for example, by their Cartesian coordinates, (x_i, y_i, z_i), $i = 1 \ldots N_{\text{ref}}$. In both subsections, for those models where an analytic solution is not known (that is, for all models except A2B8TS), we define "error" as deviation from a reference solution generated from Richardson-extrapolation of medium- and high-resolution Čadež results to the appropriate set of reference points. We remark that, in general, these points did not coincide with points actually used in the various "bare" Čadež, and Cartesian, computations. Thus, some post-processing of *all* the basic results was generally necessary before the point-wise subtractions required to produce error estimates could be performed. We are confident, however, that we have, for the most part, succeeded in keeping the error due to post-processing small in comparison to the fundamental truncation errors of the various schemes, so that the reported levels of error are genuinely indicative of the level of error in the basic solutions.

The reference points in this comparison, were simply the points used in the various low-resolution Čadež computations. In this case, extrapolated values were generated using a two-step procedure: 1) the medium- and high-resolution values were interpolated to the reference points, (interpolation was necessary due to the "zone-centered" nature of the Čadež differencing scheme), then 2) an appropriate linear combination of these interpolated values was formed to yield Richardson-extrapolated values at the reference points.

Some care is required in producing such extrapolated results. All of the extrapolations are based on the premise that the solution, $\psi^{\text{Čadež}}$, produced by the Čadež scheme, has the asymptotic ($h \to 0$) expansion

$$\psi^{\text{Čadež}} = \psi + h^2\, e_2^{\text{Čadež}} + h^4 e_4^{\text{Čadež}} + \cdots\,, \tag{5.27}$$

where h is the basic scale of discretization and $e_2^{\text{Čadež}}$, $e_4^{\text{Čadež}}$, ... are h-independent functions. In the limit $h \to 0$, we expect output from the Čadež algorithm to adhere more and more accurately to (5.27) *on the Čadež grid points.* In order to produce two-level extrapolated results at a *different* set of points, we must ensure that we interpolate the "bare" results to sufficiently high order that we preserve the first two terms of (5.27). For example, in the two-step procedure described above, *linear* ($O(h^2)$) interpolation of the medium- and high-resolution results to the low-resolution grid points would be insufficient since that procedure would introduce new $O(h^2)$ terms which, except for special mesh geometries, would not be (significantly) "cancelled" by Richardson extrapolation. In order to interpolate both ψ and $h^2\, e_2^{\text{Čadež}}$ correctly, we need to use at least *quadratic* ($O(h^3)$) interpolation. In the generation of *all* of our Richardson-extrapolated reference results we have been conservative in our interpolation and (polynomial-)extrapolation operations, invariably using a higher order of interpolation than is strictly necessary.

As mentioned above, the various "bare" results which were produced by running each code/model pair at three different resolutions also had to be postprocessed in order to produce values at the specific sets of reference points just described. The Čadež values were (again conservatively) cubically ($O(h^4)$) interpolated to the reference locations, while linear interpolation was employed for the Cartesian quantities (fundamental considerations and direct numerical experiments indicate that the $O(h^2)$ errors incurred by the linear interpolation are negligible in comparison to the truncation error of the scheme). We also note that we made no attempt to compute errors in the Cartesian results at points which lie outside the (rather small) Cartesian computational domain.

In Section 5.4.5, we quantify the basic level of error in any of the numerical computations, ψ^{model} in terms of the following discrete ℓ_1 and ℓ_∞ norms of the relative deviation from the reference solution, ψ^{ref}:

$$\|e\|_1 \equiv \frac{1}{N'_{\text{ref}}} \sum_{i=1}^{N'_{\text{ref}}} \frac{|\psi_i^{\text{model}} - \psi_i^{\text{ref}}|}{\psi_i^{\text{ref}}} \tag{5.28}$$

$$\|e\|_\infty \equiv \max_{i=1}^{N'_{\text{ref}}} \frac{|\psi_i^{\text{model}} - \psi_i^{\text{ref}}|}{\psi_i^{\text{ref}}} \tag{5.29}$$

where the index i in these expressions ranges over the N'_{ref} of the N_{ref} values which lie within the computational domain of the particular calculation under consideration. For the finite difference solutions, we also estimate a convergence rate by computing the ratio

$$\frac{\ln\left(\|e^{h_1}\|_1 / \|e^{h_2}\|_1\right)}{\ln(h_1/h_2)} \tag{5.30}$$

where h_1 and h_2 are the low and medium, or medium and high resolution discretization scales. Clearly, in the limit $h_1, h_2 \to 0$, this ratio should approach p for an $O(h^p)$ difference scheme.

5.4.5 Results of the comparison

Global error analysis

The results of the comparison are summarized in Table 5.2; here we make a few additional comments about overall features of the comparison as well as the general performance of the two methods. In the first place, and perhaps most importantly, we observe that the tabulated results show a satisfying level of agreement among the output of the schemes. For both methods, we generally have average agreement of the high resolution results to within 1% or so, and for most of the calculations, there is a general trend of decreasing deviation with increasing resolution.

In addition to this overall agreement, the superior accuracy of the Čadež results in comparison to the results from the other two methods is also striking. Particularly noteworthy is the clear $O(h^2)$ convergence manifested by all of the Čadež computations; this convergence behavior is significantly better than that of the Cartesian algorithms and, as a result, the high-resolution Čadež results are generally well over an order of magnitude (and in some instances nearly two orders) more accurate than the best Cartesian results. Furthermore, as previously discussed, because of this rather precise $O(h^2)$ convergence, the Čadež results can be substantially improved using Richardson extrapolation. For example, Table 5.2 shows that for the time-symmetric computation (A2B8TS), two-level extrapolation provides nearly a 500-fold improvement on the accuracy of the high-resolution results (the improvement in the ℓ_∞ norm is not so dramatic, but we have not studied this issue in

Table 5.1: Parameters for various two-hole calculations used in the comparison.

Model	α	β	$\vec{P}_1/a_1$	$\vec{S}_1/a_1^2$	$\vec{P}_2/a_1$	$\vec{S}_2/a_1^2$	$r_{\rm outer}/a_1$
A2B8TS	2	8	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	730
A1B8	1	8	$(14,0,0)$	$(-280,280,0)$	$(-14,0,0)$	$(0,280,280)$	350
A1B8NR	1	8	$(14,0,0)$	$(-280,280,0)$	$(-14,0,0)$	$(0,280,280)$	17
A2B8	2	8	$(15,0,0)$	$(-20,20,0)$	$(-15,0,0)$	$(0,20,20)$	730
A2B8NR	2	8	$(15,0,0)$	$(-20,20,0)$	$(-15,0,0)$	$(0,20,20)$	18

any detail). Based on the similarity of the observed levels of deviation in the Čadež results from model to model, we conclude (self consistently!) that the other extrapolated solutions (reference solutions) have similar accuracy.

In contrast to the Čadež results, the general convergence properties of the Cartesian data sets are more difficult to assess and quantify. As anticipated in the above section, we believe that the somewhat erratic convergence behavior of the method seen in Table 5.2 is largely attributable to: 1) The nature of our approximation of the inner boundary conditions (which are generally $O(h)$ accurate while the discretization of the interior equations is $O(h^2)$), and 2) the small diameter of the Cartesian computational domain. The A1B8NR and A2B8NR models were introduced specifically to estimate the relative effect of this second aspect of the Cartesian computations on the overall level of error. As can be seen from Table 5.2, errors in the Cartesian results in comparison to the NR reference solutions *are* significantly less than the deviations from the solution of the corresponding base model. In addition, the Cartesian convergence rates of the NR data sets are clearly superior for both the A1B8 and A2B8 models. For a model like A1B8 the error induced by imposing the outer boundary condition at such a small radius is a significant (if not dominant) fraction of the total error, even for the low-resolution computation.

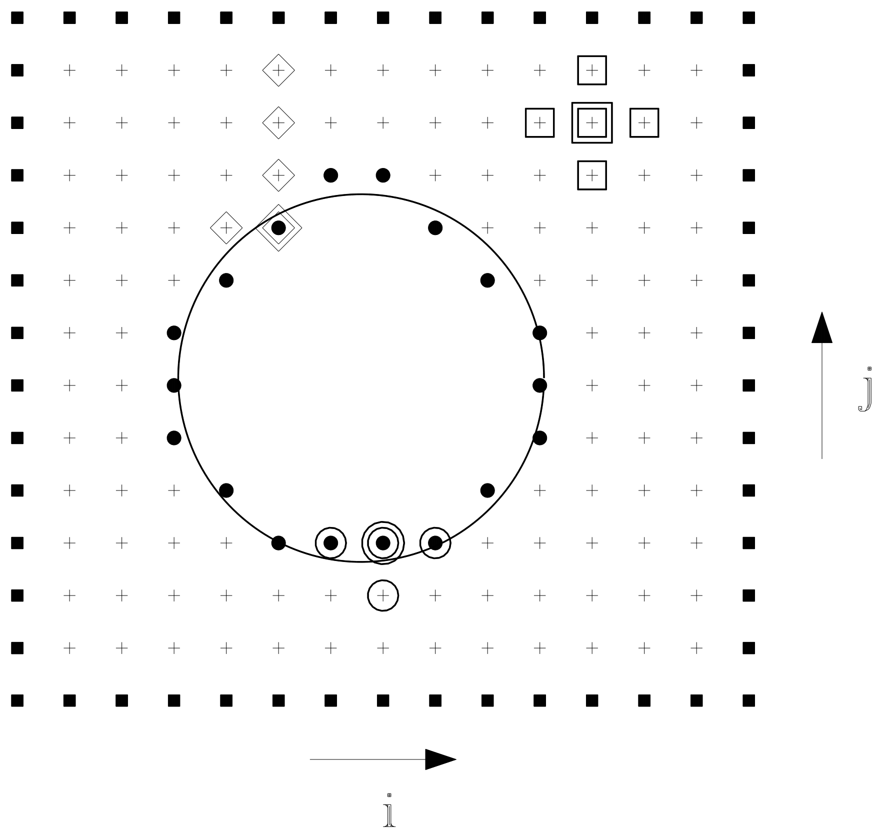


Figure 5.1: Schematic representation of the computational domain for a hypothetical, 2D, single-hole Cartesian calculation.

Table 5.2: Norms of point-wise relative deviations in ψ using extrapolated Čadež values as reference except for A2B8TS where the reference solution is analytic.

		Čadež		Cartesian	
Model	Res.	$\|e\|_1$ (conv. rate)	$\|e\|_\infty$	$\|e\|_1$ (conv. rate)	$\|e\|_\infty$
A2B8TS	low	2.41×10^{-3}	1.06×10^{-2}	3.9×10^{-2}	1.9×10^{-1}
	med.	5.84×10^{-4} (2.04)	2.67×10^{-3}	1.5×10^{-2} (2.4)	1.0×10^{-1}
	high	1.45×10^{-4} (2.01)	6.29×10^{-4}	5.1×10^{-3} (3.7)	2.5×10^{-2}
	extrap.	3.12×10^{-6}	1.33×10^{-4}	—	—
A1B8	low	6.80×10^{-4}	5.19×10^{-3}	1.3×10^{-2}	4.5×10^{-2}
	med.	1.65×10^{-4} (2.04)	1.42×10^{-3}	1.3×10^{-2} (0.0)	4.4×10^{-2}
	high	4.13×10^{-5} (2.00)	3.54×10^{-4}	1.3×10^{-2} (-0.2)	4.4×10^{-2}
A1B8NR	low	1.43×10^{-3}	5.17×10^{-3}	2.3×10^{-3}	1.2×10^{-2}
	med.	3.48×10^{-4} (2.04)	1.42×10^{-3}	2.1×10^{-3} (0.3)	1.1×10^{-2}
	high	8.71×10^{-5} (2.00)	3.54×10^{-4}	2.0×10^{-3} (0.1)	1.1×10^{-2}
A2B8	low	7.13×10^{-4}	4.88×10^{-3}	1.2×10^{-2}	7.5×10^{-2}
	med.	1.69×10^{-4} (2.08)	1.57×10^{-3}	9.8×10^{-3} (0.5)	9.0×10^{-2}
	high	4.23×10^{-5} (2.00)	3.92×10^{-4}	6.3×10^{-3} (1.5)	2.0×10^{-2}
A2B8NR	low	1.74×10^{-3}	4.93×10^{-3}	8.9×10^{-3}	7.4×10^{-2}
	med.	4.12×10^{-4} (2.07)	1.58×10^{-3}	6.1×10^{-3} (0.9)	8.8×10^{-2}
	high	1.03×10^{-4} (2.00)	3.96×10^{-4}	2.7×10^{-3} (2.8)	1.4×10^{-2}

Chapter 6

Test problems

In this chapter we implement most of the tools that we will need in designing a 3D extrapolatable MLAT code for the initial value problem of 2 black holes. A complete code has not yet been implemented. The original tools we implemented in this chapter are in sections (6.3), (6.4), (6.5), (6.6), and (6.7). We go through a series of test problems each usually implementing one new technique which is different from an (all-Dirichlet) PDE, which we refer to as the "model problem".

The first problem, found in Section 6.1 compares a basic 3D multigrid code with a SOR code, a Gauss-Seidel code, and a Jacobi code. We will clearly see that the multigrid algorithms are the most efficient algorithms for solving elliptic systems[12][20] . We also boost the accuracy of the solution using Richardson extrapolation, and show how this can easily be introduced in the multigrid algorithm[12][20].

The second problem, found in Section 6.2, tests the adaptive routines which will be used in our 3D multigrid code for 2 black holes. Here we assume an analytic solution:

$$\psi \approx 1 + \frac{a_1}{r_1} + \frac{a_2}{r_2}$$

which is a good approximation for ψ for two black holes. This code will estimate the truncation errors on a lattice and then refine the regions where it determines that the truncation errors are large . This code will then generate grid structures of the type which eventually will be used in a complete MLAT algorithm for the initial value problem.

The third problem, found in Section 6.3, is concerned with the outer boundary condition, equation (2.34), and we design a 1D multigrid code using this boundary condition. Here we use a first order stencil, and cycle using deferred correction. We see here that the outer boundary is easy to implement. We also demonstrate that in order to Richardson extrapolate, we must use deferred correction to an order greater than 3.

The fourth problem, found in Section 6.4, examines the issue of Richardson extrapolation in a MLAT code. Here we find empirically a method which generates extrapolatable solutions in MLAT codes. Our solution to this problem involves smoothing the truncation error estimates around grid interfaces. Here we consider both one-dimensional and two-dimensional problems and design multigrid codes for both. We see that that the results from both codes can be Richardson-extrapolated.

The fifth problem, found in Section 6.5, considers the difficulties encountered in designing an efficient multigrid code in one dimension, when the grids are non-contained. Here we must determine the correct transfers for the boundary equations, and for the interior equations, whose points are close to the boundaries. We consider this issue since a three-dimensional multigrid code for the treatment of a problem on an unbounded physical domain will not have a uniform computational domain, *i.e.* the coarsest grid will certainly not cover the entire domain, nor will the finest.

The sixth problem, found in Section 6.6, combines features from the previous three problems in the construction of a general 1 dimensional multigrid code with inner and outer boundary conditions. Since derivatives can not be approximated well on very coarse grid, we would like to use a Dirichlet condition for the inner boundary on the coarser grids, and then use a mixture of a Dirichlet condition and the Robin condition on intermediate grids. Thus, we use a blended boundary condition implemented on the inner boundary, to help improve the convergence rate. Finally, on the finest levels, we will apply the true Robin condition.

For the last problem, found in Section 6.7, we implement a two dimensional multigrid code on a domain like the one shown in Figure 6.46, where the points inside the hole are not in the computational domain. We use special smoothing around the boundary of the hole, along with deferred correction.

By using more the more powerful techniques developed in this chapter we should be able to make a new version of `genpsi` which not only outperforms the old version in speed, but also in accuracy. We should also be able to use these algorithms in designing other codes which will be used to solve other elliptic equations in numerical relativity. Here we have a cleaner mathematical formulation of our problem than might usually be the case and our boundary geometries, though non-trivial, are nowhere near as intricate as those often encountered in engineering applications (airfoils etc.). We note another reason people have not used extrapolation etc. is that it is more work and you do not get better results until you already have solutions which are generally accurate enough in any case. In our case the advantage of extrapolating low-order results lies in the need for extremely high accuracy for wave prediction, which justifies the effort.

6.1 Four iterative algorithms to solve a model problem in three dimensions

In this section, we wish to solve a model elliptic problem:

$$\frac{\partial^2 u\left(x,y,z\right)}{\partial x^2} + \frac{\partial^2 u\left(x,y,z\right)}{\partial y^2} + \frac{\partial^2 u\left(x,y,z\right)}{\partial z^2} = f\left(x,y,z\right) \tag{6.1}$$

with Dirichlet boundary conditions, where $x \in (0 \cdots 1)$, $y \in (0 \cdots 1)$ and $z \in (0 \cdots 1)$.

This is finite differenced on a uniform mesh (constant mesh spacing h in all three directions) to $O\left(h^2\right)$ as usual:

$$\begin{aligned} h^{-2}\left(u[j+1,k,l] + u[j-1,k,l] + u[j,k+1,l] + u[j,k-1,l]\right) \\ +h^{-2}\left(u[j,k,l+1] + u[j,k,l-1] - 6u[j,k,l]\right) = f[j,k,l]. \end{aligned} \tag{6.2}$$

We then define $f[j,k,l]$ so that the solution is

$$u[j,k,l] = \sin\left(2\pi x[j]\right) \times \sin\left(2\pi y[k]\right), \times sin\left(2\pi z[l]\right)$$

i.e.;

$$f[j,k,l] = -6\pi^2\left(\sin\left(2\pi x[j]\right) \times \sin\left(2\pi y[k]\right) \times sin\left(2\pi z[l]\right)\right). \tag{6.3}$$

which defines the boundary Dirichlet values.

In this section we compare the CPU time, which was measured on a CRAY YMP, the work unit,the final ℓ_2 norm of the residuals, the convergence rate, and the error for four iterative algorithms.

6.1.1 The algorithms

The Jacobi algorithm we use is a point-wise algorithm shown in Figure 6.1. The Gauss-Seidel and SOR codes have similar algorithms, but the `uold`'s in the Jacobi algorithm are replaced by `unew`'s and an ω factor is placed inside the algorithm, setting $\omega = 1$ for the Gauss-Seidel code. The algorithms are shown in Figure 6.2, where we use ω_{optimum} which was defined in chapter 3.

The full multigrid algorithm uses the same algorithm defined in Figure 4.11, which we also include in Figure 6.3 for convenience. The smoother is defined by the `RELAX` routine.

A general cubic interpolation routine[14] is used to transfer the solution of a coarse grid to a fine grid, on all grids which have over 5 points per side. We find that cubic interpolation does help to reduce the residuals by an order of magnitude over linear interpolation. In our test we saw that v cycles were the fastest type of cycle. We also found empirically

Figure 6.1: The Jacobi algorithm.

```
PROCEDURE JACOBI( l, res2)
Solve the model problem
   Initialize_grid_variables(l) with nx,ny,nz points
   n = 0    No iterations have taken place
   START
      n = n + 1
      Relax the system via equation (3.48)
      uold = unew
      IF(> res2) GOTOSTART
   Compute the error
END PROCEDURE
```

Figure 6.2: The SOR/Gauss-Seidel algorithm

```
PROCEDURE SOR/GAUSS-SEIDEL ( l, res2)
Solve the model problem
   Initialize_grid_variables(l) with nx,ny,nz points
   n = 0    No iterations have taken place
   START
      n = n + 1
      Relax the system via equation (3.61)
      IF(> res2) GOTOSTART
   Compute the error
END PROCEDURE
```

that multigrid performance was best when `pre` and `pst` the number of relaxation sweeps performed before and after a coarse grid correction respectively, were both set to 3. We also used linear interpolation and half-weighted restriction (which we found superior to full-weighted restriction).

6.1.2 Results

In this section we present the performance of the above algorithms on a model problem. We run each of these codes for levels 2 through 7, where the number of points on a cube edge, n is defined by

$$n = 2^{\mathrm{l}} + 1. \tag{6.4}$$

The `TIME` is the total computational time, measured in seconds, W is the work units (recall one work-unit is the computational work needed to relax on the finest grid) divided by 25.8, which normalizes the FMG FAS work to unity, and ρ is the convergence rate, defined in

Figure 6.3: The FAS FMG algorithm.

```
PROCEDURE FullMultigrid_FAS( l_max, ncyc, pre, pst )
   Initialize Variables
   Initialize Memory
    SOLVE(L^H u^H = f^H)     Solve the coarsest level problem
   DOl = 1 , l_max
      DOcycle= 1 , ncyc
       u^h := II_H^h u^H
      VCycle_FAS( cycle, l , pre, pst)
      END DO   for cycle
   END DO   for l
END PROCEDURE

PROCEDURE VCYCLE_FAS( cycle, l_max, pre, pst )
    Cycle up to the coarsest level
    DOl = l_max, l_coarse + 1
      DOp = 1 , pre
          RELAX(u^h = f^h)     Smooth the residuals
      END DO
       tau_h^H = L^H I_h^H u^h - I_h^H L^h u^h  determine the truncation error estimate
       f^H = I_h^H f^h + tau_h^H     Correct the rhs
    END DO

   Now solve the system on the coarsest level
    SOLVE(L^H u^H = f^H)
   Now come back down to the finest level, performing the needed corrections
   DOl = l_coarse, l_max - 1
       Determine the correction
       u^h := u^h + I_H^h (u^H - I_h^H u^h)
       Now smooth out u^h
      DOq = 1 , pst
         RELAX(u^h = f^h)     Smooth the residual
      END DO
    END DO
 END PROCEDURE
```

Section 4.5, as

$$\rho \equiv W^{-1} \log_{10} \frac{||r^{\text{initial}}||}{||r^{\text{final}}||} \tag{6.5}$$

We also represent the ℓ_2 norm of the relative error as e.

We see that the multigrid code typically takes about 4 relaxation sweeps on the finest grid in order for the residuals to be reduced by one order of magnitude. There are two other plots of a type which we will often show in later chapters. The first is a plot of the ℓ_2 norm of the residuals as a function of W (the work unit) number. Figure 6.4 shows $\log_{10}$ of the ℓ_2 norm of the residuals versus the work for the SOR code, the Gauss-Seidel code, and the multigrid code. This data was generated from the level 6 run. The other plot, Figure 6.5 shows the reduction of the error, e, versus the work unit, W. This figure clearly shows that in the SOR algorithm, over 200 unnecessary sweeps were done, since we can easily see that after about 250 work units, the error is not reduced any more. The multigrid code which we used did not use an adaptively determined stopping criterion of the type discussed in Chapter 4 and we see that we could have easily reduced the work units on the multigrid code too. We also see that the Gauss-Seidel code is extremely inefficient for solving elliptic equations[12] [10]. We once again repeat that there is nothing new in these results, we include them here primarily as motivation for using multigrid. In the tables that we use from this point forward, we frequently represent numbers as $a(b)$ which is equivalent to $a \times 10^b$.

Table 6.1: 3D FMG FAS Multigrid algorithm results

l	TIME	W	Final r_2	ρ	e
2	0.01	0.8	1.2(-13)	-7.5(-1)	5.9(-2)
3	0.02	1.0	8.5(-9)	-0.36	1.6(-2)
4	0.08	1.1	3.4(-7)	-0.26	4.2(-3)
5	0.33	1.0	2.8(-7)	-0.25	1.1(-3)
6	1.63	1.0	8.6(-8)	-0.25	2.8(-4)
7	9.43	1.0	2.3(-8)	-0.25	7.0(-5)

Table 6.2: 3D SOR algorithm results

l	TIME	W	Final r_2	ρ	e
2	0.02	1.5	1.0(-12)	-3.5(0)	5.9(-2)
3	0.04	2.2	5.8(-9)	-1.7(-1)	1.6(-2)
4	0.18	3.9	2.8(-7)	-8.4(-2)	4.2(-3)
5	1.43	7.7	2.1(-7)	-4.3(-2)	1.1(-3)
6	15.51	16.7	3.2(-8)	-2.2(-2)	2.8(-4)
7	194.00	34.6	1.6(-8)	-1.1(-2)	7.0(-5)

Table 6.3: 3D Gauss-Seidel algorithm results

l	TIME	W	Final r_2	ρ	e
2	0.01	0.1	1.0(-13)	-7.2(+1)	5.9(-2)
3	0.03	1.6	4.3(-9)	-2.9(-1)	1.6(-2)
4	0.21	4.8	3.1(-7)	-6.7(-2)	4.2(-3)
5	3.50	19.5	2.8(-7)	-1.7(-2)	1.1(-3)
6	75.80	83.1	8.6(-8)	-4.1(-3)	2.8(-4)
7	1866.88	333.1	2.3(-8)	-9.8(-4)	7.0(-5)

Table 6.4: 3D Jacobi algorithm results

l	TIME	W	Final r_2	ρ	e
2	0.01	0.1	1.0(-12)	-7.2(+1)	5.9(-2)
3	0.05	2.5	8.2(-09)	-1.5(-1)	1.6(-2)
4	0.45	9.1	3.2(-07)	-3.4(-2)	4.2(-3)
5	6.70	37.6	2.8(-07)	-8.4(-3)	1.1(-3)
6	137.10	150.2	8.6(-08)	-2.1(-3)	2.8(-4)
7	3376.21	602.4	2.3(-08)	-5.2(-4)	7.0(-5)

Figure 6.4: A plot of the ℓ_2 norm of the residual versus the work unit for a six level run

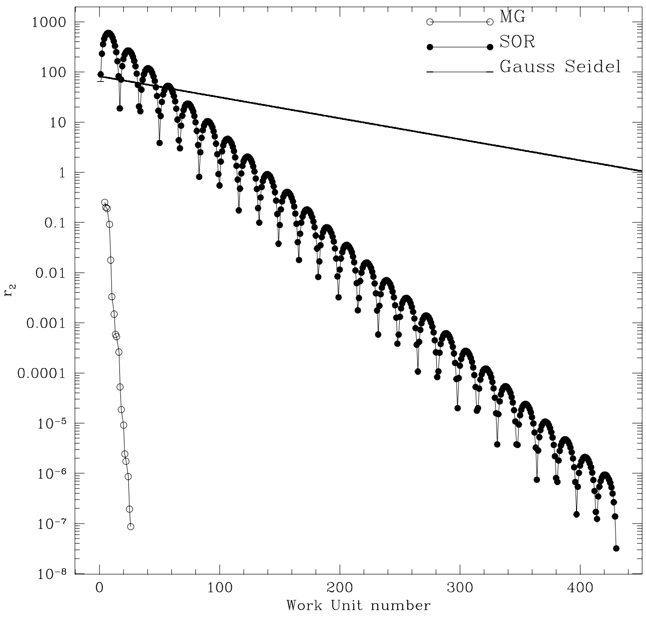
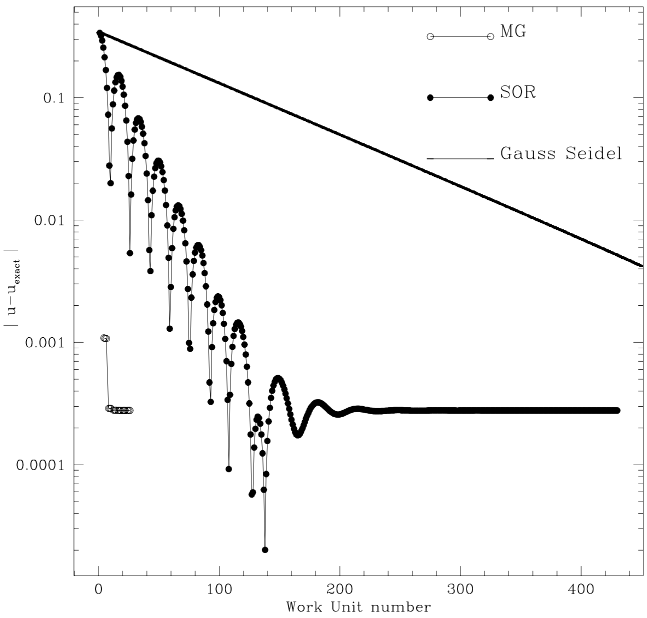


Figure 6.5: A plot of the absolute error versus the work unit for a six level run.



6.2 Adaptive tests

The purpose of this section is to see how well grid adaptation of the type discussed in Section 4.10 will work with the type of function which tends to approximate ψ. A good approximation to ψ for two black holes with zero extrinsic curvature is

$$\psi_{model} = 1 + \frac{a_1}{r_1} + \frac{a_2}{r_2}. \tag{6.6}$$

Bowen & York[7] also suggest a model ψ for one hole as

$$\psi_{model} = \left(1 + \frac{\alpha}{r} + \frac{\beta}{r^2} + \frac{\gamma}{r^3} + \frac{\delta}{r^4}\right)^{\frac{1}{4}} \tag{6.7}$$

where α, β, γ and δ are constants. We use a model ψ with a similar form:

$$\psi = \alpha + \frac{\beta}{r_1} + \frac{\gamma}{r_2} + \frac{\delta}{r_1^2} + \frac{\epsilon}{r_2^2} + \frac{\eta}{r_1 r_2}.$$

where $r_1 = r - c_1, r_2 = r - c_2$, and α, β, γ, δ, ϵ, and eta are constants.

We use this ψ to determine f from

$$f = \nabla^2 \psi. \tag{6.8}$$

We then compute the local truncation error,τ^h,via equation (4.27),

$$\tau^h \equiv L^h \psi - f^h.$$

Finally we compare τ^h with $\tau_{\max}$ to see where the grid needs to be refined, and we then refine the grid at this level, thereby generating one or more new grids. We will fully document the algorithms in the sections below, beginning with the one dimensional case.

6.2.1 Adaptive Mesh Refinement (AMR) in 1D

The 1D code is described fully by the algorithm in figures (6.6) , (6.7), and (6.8) . Here `sg2` is an input parameter which, if equal to one, specifies that each grid must have its total number of points equal to $2^n + 1$, where n is an integer.

We now test the routines using the fixed input parameters: $x_{\min} = -100$, $x_{\max} = 100$, $\tau_{\max} = 0.01$ and $l_{\max} = 20$. We show the bounding limits of each grid, (Figure 6.17), by generating a rectangle, which has a height equal to half of a level. The bottom of the rectangles are at the level of the grid. The input parameters of these runs are shown in Table 6.5.

The output of this code generates a graph and listing. The output of the first run is shown in Table 6.6, where we include output only for the first 10 levels. The corresponding graph is shown in Figure 6.15

A careful look at this output shows that the grid refinements are "zooming" into the singular points. In fact, on the finest level, we are using over 5000 points. The other noticeable thing about this run is that, except at the coarsest level, each grid covers a smaller region than its parent. Run 2 differs from run 1 only in that we demand that the number of points in a grid must be $2^n + 1$ for some n. We see from the results in Table 6.7 that there are many levels where `GXMIN` and `GXMAX` do not change from their parents. Thus, there is much waste in this refinement, and in general we *should not* be concerned with having children maintain `NX` $= 2^n + 1$.

This one dimensional test is not a "true" test of the problem which we will be trying to model in higher dimensions. Since the computational domain will end at approximately the radius of each hole, the grids will never approach the singularity. Thus, our multidimensional test cases will have the option of whether or not to stop the domains at or near the edge of the hole boundaries.

The third run was made such that we would be able to compare the grid refinements of a run where the higher order terms in ψ were small compared to the first two runs. The results for this test are shown in Table 6.2.3 and Figure 6.17. It is extremely difficult to tell the difference between the graphs of these two runs, thus, we should also look at tables (6.6) and (6.2.3). We see that at the 10^{th} level, there is only about half the number of points being used on this run.

We have also built the foundation for the multidimensional codes. Building one dimensional codes before the multidimensional cases appears to be an excellent practice, since the one dimensional cases can be implemented, tested and evaluated much more rapidly than their multidimensional counterparts.

6.2.2 AMR in 2D

The 2D code is described fully by the algorithm shown in figures (6.9), (6.10), (6.11), (6.12), (6.13) and (6.14).

We now test the routines using the fixed input parameters:
$x_{\min}, y_{\min} = -100$, $x_{\max}, y_{\max} = 100$, $\tau_{\max} = 0.0005$ and $l_{\max} = 14$ and `sg2 = 0`. The input parameters of this run are shown in Table 6.8.

The output of the first run is shown in Table 6.9 although we truncated the output to show only the first seven levels. To make the graphs easier to view, we first outline a bounding box for each grid. We then shade each grid according to its level, the finest level of each run is black, whereas the coarsest level is white. We also include figures which zoom in on certain regions for each graph. The graphs of this run are shown in figures (6.18), (6.19), and (6.20). As we expect, the finest grids form a circle around the radius of the smallest hole, since the truncation error estimates get greater as we approach the singularity. In this run, we allowed 14 maximum levels, but the run stopped at 10 levels since $\tau[j,k] < \tau_{\max}$ at level 11. We also see from the graphs of these runs, that our clustering algorithm does not work extremely well. Berger and Rigoutsos[3] have a much better algorithm which we plan to use in the future. The truncation error must be isotropic since both of the "unknowns" (u) and differential operator are. Our algorithm encounters difficulties when trying to resolve the regions around the holes since not all of the grids overlap.

The only difference between the first two runs, is that the second run *does not* set $\tau[j,k] = 0$ inside the holes. Thus, we should expect the grids to zoom into the singularity. Since we do not allow any grid to have more than 512×512 points, the run stopped at the 10^{th} level although the truncation errors were not below $\tau_{\max}$. Once again we show three graphs of the bounding boxes of the grids for this run. These graphs are shown in figures (6.21), (6.22), and (6.23).

In the third run, we let the centers of the two holes be very far from one another. We also let the radii of both holes be equal to 1/2. This run required 13 levels of refinement to get the truncation error values below $\tau_{\max}$. The graphs are shown in figures (6.24), (6.25) and (6.26). We see that as soon as we got to the fourth refinement level, the grids were split into 2 regions. We see that the grids are symmetric from one hole to the other as would be expected. The third image for this run zooms into one of the holes to see the actual refinement grids used around the hole. Once again we see that the clustering routine has done a poor job of refining the region around the finest level. Before we can incorporate our clustering routine into our 3D multigrid code, we must improve its performance.

We can expect that these routines can be used as regridding modules in our 3D MLAT code. Of course that MLAT code will also use a relative truncation error estimate instead of the true truncation error. We did not worry about generating a table of the efficiencies since our algorithm does not work extremely well. As we stated before, the finest grids around a hole do not always overlap, which is a extreme deficiency in the algorithm.

6.2.3 AMR in 3D

The 3D routines are almost identical to the 2D routines. The only real difference between the two versions is in the clustering routines. In particular, in the procedure `PROCEDURE CLUSTER_1` we now add two more ways to look, `UP` and `DOWN` which makes the routine search in a 7 point stencil now. Since it is extremely difficult to display the grid structure, we do not, particularly since the results look very similar to the two-dimensional case.

Figure 6.6: The AMR algorithm

```
PROCEDURE ADAPT_GRIDS_1D( α,β,γ,δ,ε,η, c1, c2,min, max
                                    τmax,sg2,lmax )
   Program determines τ^h from ∇²u^h = f^h
   where u^h = α + β r1^-1 + γ r2^-1 + δ r1-2 + ε r2^-2 + η (r1 r2)^-1
   h = 1/2 (max - min)    coarsest grid has 3 points
   DEFINE_BASE_GRID( 1 , h, sg2, min, max, gxmin , gxmax , nx )
   DEFINE_GRID_FUNCTIONS( lhead(0) , nx, gxmin, gxmax ,h,
                                    α,β,γ,δ,ε,η,x,u,f,τ)
   l = 0
   refine =  TRUE
   grid = lhead(l))
   Start to loop on all the grids at this level, by first finding
   the head of the level, and then finding all of the neighbors
   START
      IF refine = FALSE GOTOEND
      bins = CLUSTER_GRID( grid, nx, sg2, τ , τmax, imin, icount )
      The routine CLUSTER_GRID , makes bins clusters of points with imin(bins))
      used to store the starting values and icount(bins) storing
      the number of points in the bin
      IF( bins > 0) GENERATE_NEW_GRIDS( grid, bins, imin, icount,sg2,
                                         gxmin, gxmax,nx )
      The routine GENERATE_NEW_GRIDS determines the number of points on
      the grid and the extent of the grid. The routine also sets up
      all the new linked lists
      DEFINE_GRID_FUNCTIONS( grid , nx, gxmin, gxmax ,h,
                                    α,β,γ,δ,ε,η,x,u,f,τ)
      Now DEFINE_GRID_FUNCTIONS will set up all the grid
      functions on all children of grid grid
      IF(bins > 0) refine =  TRUE
      grid = gpngh(grid)
      Set grid equal to its neighbor grid
      GOTOSTART
   END
   PRINT GRIDS
END PROCEDURE  .
```

Table 6.5: 1D AMR test parameters

run number	sg2	α	β	γ	δ	ϵ	η	c_1	c_2
1	0	1.0	1.5	1.0	10.0	10.0	0.0	-10.0	10.0
2	1	1.0	1.5	1.0	10.0	10.0	0.0	-10.0	10.0
3	0	1.0	1.0	1.0	0.0	0.0	0.0	-10.0	10.0

Figure 6.7: The AMR algorithm to define the base grid

PROCEDURE DEFINE_BASE_GRID(grid, h, sg2, min, max, gxmin, gxmax, nx)
This procedure will set up gxmin ,gxmax and nx for the grid
   IF(sg2 = 0) THEN
      gxmin(grid) = min
      gxmax(grid) = max
      nx = 1 + h^{-1} (gxmax(grid) $-$ gxmin(grid))
   ELSE
      We will only use $2^n + 1$ points on this gridi
      nx = $1 + 2^{\log_2(h^{-1}(max-min))}$
      gxmin(grid) = min
      gxmax(grid) = gxmin(grid) + h * (nx - 1)
   END IF
END PROCEDURE .

PROCEDURE DEFINE_GRID_FUNCTIONS(grid , nx, gxmin, gxmax , $h, \alpha, \beta,$
$\gamma, \delta, \epsilon, \eta, x, u, f, \tau$)
This routine will determine x, u, f, and τ for the grid
   GETX(nx, gxmin(grid), h, x)
   GETU(nx, $x, \alpha, \beta, \gamma, \delta, \epsilon, \eta, u$)
   GETF(nx, $x, \alpha, \beta, \gamma, \delta, \epsilon, \eta, f$)
   GET_TAU(nx, h, u, f, τ)
END PROCEDURE

PROCEDURE GENERATE_NEW_GRIDS(grid, bins, imin, icount, sg2,
gxmin, gxmax, nx)
This procedure will free all the memory to be used with this grid along with setting up all the links for lhead , gpp , gpngh , gpsib ,*and* gpch.
   DOi = 1 , bins
      g=llfree(llnext)   *g is the next grid number that has not been used*
      gl(g) = gl(grid) + 1
      min = x(imin(i))
      max = x(imin(i) + icount(i) $-$ 1)
      DEFINE_BASE_GRID(grid, h, sg2, min, max, gxmin, gxmax, nx)
      lhead(g) = g
      gpp(g) = grid
      gpch(g) = UNDEF
      gpsib(g) = find_sib(g)
      gpngh(g) = find_ngh(g)
   END DO
END PROCEDURE

Figure 6.8: The 1D clustering AMR algorithm.

```
PROCEDURE CLUSTER_GRID(grid, nx, sg2, τ , τ_max, imin, icount )
This functions returns the number of unique group of points
   τ(1) = τ(2)     τ is not defined on end points
   τ(nx) = τ(nx − 1)
   goodbins = 0
   flag = FALSE
   Now loop over all points and separate flagged points into clusters
   DOi = 1 , n
      IF(flag  AND ABS(τ(i)) ≥ τ_max ) THEN
         goodbins = goodbins + 1
         minbin(goodbins) = i
         flag = FALSE
      ELSEIF( NOT flag  AND ABS(τ(i)) < τ_max) THEN
         flag =  TRUE
         maxbin(goodbins) = lastx
      ELSE
         lastx = i
         IF( NOT flag  AND goodbins > 0) THEN
            maxbin(goodbins) = lastx
         END IF
      END IF
   END DO
   Now all the points are flagged and separated into unique bins
   Now we must make sure that each bin contains an odd number of points
   DOi = 1 , goodbins
      IF( MOD( (maxbin(i) - minbin(i)),2 ) NOT= 0 ) THEN
         minbin(i) = minbin(i) + 1
         We just add an extra point to the front of the bin
      END IF
   END DO
```

Table 6.6: 1D AMR test, output for run1

l	GXMIN	GXMAX	NX
0	-1.00000e+02	1.00000e+02	3
1	-1.00000e+02	1.00000e+02	5
2	-5.00000e+01	5.00000e+01	5
3	-2.50000e+01	2.50000e+01	5
4	-2.50000e+01	2.50000e+01	9
5	-2.50000e+01	2.50000e+01	17
6	-1.56250e+01	1.56250e+01	21
7	3.12500e+00	1.56250e+01	17
7	-1.56250e+01	-3.12500e+00	17
8	-1.40625e+01	-6.25000e+00	21
8	6.25000e+00	1.40625e+01	21
9	6.25000e+00	1.32813e+01	37
9	-1.32813e+01	-6.25000e+00	37
10	-1.28906e+01	-7.42188e+00	57
10	7.22656e+00	1.26953e+01	57

Figure 6.8 continued

```
    Now merge together any bins that overlap
    total_bins = MERGE_BINS(goodbins,minbin,maxbin,x_min,x_max)
    This routine will return the total number of unique bins = MERGE_BINS
    This routine also returns the starting points
    and ending points of the bins in x_min and x_max
    IF( sg2 =  TRUE) THEN
       START1
       DOi = 1 , total_bins
          points = x_max(i) - x_min(i) + 1
          If points is not a power of two then add points to x_min and x_max
       END DO
       new_bins = MERGE_BINS(goodbins,minbin,maxbin,x_min,x_max)
       DOi = 1 , new_bins
          points = x_max(i)- x_min(i) + 1
          If points is not a power of two THEN GOTO START
       END DO
    total_bins = new_bins
    END IF
    Now just put the total number of bins in CLUSTER_GRID
    and the total number of points in x_max(i)
    CLUSTER_GRID = total_bins
    DOi = 1 , CLUSTER_GRID
       x_max(i) = x_max(i) - x_min(i) +1
    END DO
END PROCEDURE
```

1D AMR test, output for run2

Table 6.7: 1D AMR test, output for run2

l	GXMIN	GXMAX	NX
0	-1.00000e+02	1.00000e+02	3
1	-1.00000e+02	1.00000e+02	5
2	-1.00000e+02	1.00000e+02	9
3	-5.00000e+01	5.00000e+01	9
4	-5.00000e+01	5.00000e+01	17
5	-2.50000e+01	2.50000e+01	17
6	-2.50000e+01	2.50000e+01	33
7	3.12500e+00	1.56250e+01	17
7	-1.56250e+01	-3.12500e+00	17
8	-1.56250e+01	-3.12500e+00	33
8	3.12500e+00	1.56250e+01	33
9	3.12500e+00	1.56250e+01	65
9	-1.56250e+01	-3.12500e+00	65
10	-1.32813e+01	-7.03125e+00	65
10	7.03125e+00	1.32813e+01	65

Figure 6.9: A 2D AMR algorithm

```
PROCEDURE ADAPT_GRIDS_2D( α,β,γ, c1, c2,minx, maxx,miny,maxy,
                                radius1,radius2,τmax,sg2,
                                lmax,option)
   DEFINE_BASE_GRID2D( )
                          gxmin , gxmax , gymin, gymax , nx ,ny)
   DEFINE_GRID_FUNCTIONS2D( )
   l = 0
   refine =  TRUE
   grid = lhead(l)
   START
      IF refine = FALSE GOTOEND
      bins = CLUSTER_GRID2D( grid, nx, ny, sg2, τ , τmax,
                             imin, icount,jmin,jcount )
      The routine CLUSTER_GRID , makes bins clusters of points with
      imin(bins) and jmin(bins) used to store the starting values and
      icount(bins),jcount(bins)  storing the number of points in the bin
      IF( bins > 0) THEN
         GENERATE_NEW_GRIDS2D( )
      The routine GENERATE_NEW_GRIDS2D determines the number of points
      on the grid and the extent of the grid. The routine also sets up
      all the new linked lists
      IF(bins>0) THEN
         DEFINE_GRID_FUNCTIONS2D( )
         Now DEFINE_GRID_FUNCTIONS2D will set up all the grid
         functions on all children of grid grid
         efficiency = CALC_EFF( )
         CALC_EFF computes the efficiency on the grid as described in the text
      END IF
      IF(bins > 0) refine =  TRUE
      grid = gpngh(grid)
      Set grid equal to its neighbor grid
      GOTOSTART
   END
   PRINT GRIDS
END PROCEDURE  .
```

Figure 6.10: The 2D AMR algorithm used to define the base grid

PROCEDURE DEFINE_BASE_GRID2D(grid, h, sg2, $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$
gxmin, gxmax, gymin, gymax, nx, ny)

This procedure will set up gxmin ,gxmax,gymin,gymax,nx and ny for the grid

```
   IF( sg2 = 0) THEN
      gxmin(grid) = x_min
      gxmax(grid) = x_max
      gymin(grid) = y_min
      gymax(grid) = y_max
      set nx
      set ny
   ELSE
      We will only use 2^n + 1 points on this grid
      nx = 1 + 2^(log_2(h^-1 (x_max - x_min)))
      ny = 1 + 2^(log_2(h^-1 (y_max - y_min)))
      gxmin(grid) = gxmin(grid)
      gxmax(grid) = gxmin(grid) + h * (nx - 1 )
      gymin(grid) = gymin(grid)
      gymax(grid) = gymin(grid) + h * (ny - 1 )
   END IF
END PROCEDURE .
```

PROCEDURE DEFINE_GRID_FUNCTIONS2D( )

This routine will determine x, y, u, f, and τ for the grid

option will tell whether or not τ will be defined inside radius1 *and* radius2

```
   GETX(nx,gxmin(grid),h, x)
   GETX(ny,gymin(grid),h, y)
   GETU(nx,ny,x,y,α,β,γ,c_1,c_2,radius1,radius2,u)
   GETF(nx,ny,x,y,α,β,γ,c_1,c_2,radius1,radius2,f)
   GET_TAU(nx, ny, h,u,f,option, radius1, radius2, c_1,c_2,τ)
   when option = 1 then τ(i,j) = 0 inside the two holes
END PROCEDURE
```

Figure 6.11: The 2D AMR algorithm used to generate new grids.

```
PROCEDURE GENERATE_NEW_GRIDS2D( grid, bins, imin, icount,
                                       jmin, jcount, sg2,gxmin,gxmax,
                                       gymin, gymax, nx , ny)
This procedure will free all the memory to be used with this grid
along with setting up all the links for lhead , gpp , gpngh , gpsib ,and gpch.
   DOi = 1 , bins
      g=llfree(llnext)    g is the next grid number that has not been used
      gl(g) = gl(grid) + 1
      minx = x(imin(i))
      maxx = x(imin(i) + icount(i) - 1)
      miny = y(jmin(i))
      maxy = y(jmin(i) + jcount(i) - 1)
      DEFINE_BASE_GRID( grid,h,sg2,minx,maxx,miny,maxy,
                              gxmin,gxmax,gymin,gymax,nx,ny)
      lhead(g) = g
      gpp(g) = grid
      gpch(g) = UNDEF
      gpsib(g) = find_sib(g)
      gpngh(g) = find_ngh(g)
   END DO
END PROCEDURE

PROCEDURE CLUSTER_GRID2D( grid, nx, ny, sg2, τ , τmax,
                              imin, icount,jmin,jcount )
   STEP1: First set τ(i,j) on the end points to the value one point over
   STEP2: COMPUTE char(i,j)
   DOj = 1 , ny
      DOi = 1 , nx
         char(i,j) = FALSE
         IF( ABS( τ(i,j) )≥ τmax ) char(i,j) =  TRUE
      END DO
   END DO
   STEP3: NOW form the bins
   CLUSTER_GRID2D = FORM_BIN( nx , ny, char, temp, sg2, imin,
                                    icount,jmin,jcount
END PROCEDURE
```

Figure 6.12: The 2D AMR algorithm to cluster the points.

```
PROCEDURE FORM_BIN( nx , ny, char, temp, sg2, imin, icount,jmin,jcount)
   set the temp array equal to 0 on all elements
   temp(i,j) = 0
   bins = CLUSTER_1( nx,ny,char, temp )
   FORM_RECT( nx,ny,bins, temp, imin, jmin, icount, jcount)
   IF(sg2=1) RESIZE_RECT( nx,ny,bins ,imin,jmin,icount,jcount)
   FORM_BIN = bins
END PROCEDURE

PROCEDURE CLUSTER_1( nx,ny,char, temp )
   uses a recursive function to cluster all points that are
   connected together by their nearest neighbors
   bin = 0
   RIGHT = 0
   LEFT = 1
   UP = 2
   DOWN = 3
   DOj = , ny
      DOi = , nx
         IF( CHAR(i,j)  AND (temp(i,j)=0) ) THEN
            bin = bin + 1
            temp(i,j) = bin
            IF( char(i+1,j)  AND temp(i+1,j)=0) THEN
               id = FILL( RIGHT , bin, i, j, char, temp, nx , ny)
            END IF
            IF( char(i-1,j)  AND temp(i-1,j)=0) THEN
               id = FILL( LEFT , bin, i, j, char, temp, nx , ny)
            END IF
            IF( char(i,j+1)  AND temp(i,j+1)=0) THEN
               id = FILL( UP , bin, i, j, char, temp, nx , ny)
            END IF
            IF( char(i,j-1)  AND temp(i,j-1)=0) THEN
               id = FILL( DOWN , bin, i, j, char, temp, nx , ny)
            END IF
         END IF
      END DO
   END DO
END PROCEDURE
```

Figure 6.13: The 2D AMR recursive routine to determine which points to cluster together.

```
RECURSIVE PROCEDURE FILL ( direction, bin, i_in, j_in, char, temp,nx,ny)
   This routine is capable of calling itself, and will look in all directions to
   find where char is  TRUE
   RIGHT = 0
   LEFT = 1
   UP = 2
   DOWN = 3
   i = i_in
   j = j_in
   IF(direction = RIGHT) i = i + 1
   IF(direction = LEFT) i = i - 1
   IF(direction = UP) j = j + 1
   IF(direction = DOWN) j = j - 1
   fill = 0
   IF( NOT CHARM OR temp(i,j) ≠ 0 ) THEN
      END PROCEDURE
   ELSE
      temp(i,j) = bin
      IF( char(i+1,j)  AND temp(i+1,j)=0) THEN
         id = FILL( RIGHT , bin, i, j, char, temp, nx , ny)
      END IF
      IF( char(i-1,j)  AND temp(i-1,j)=0) THEN
         id = FILL( LEFT , bin, i, j, char, temp, nx , ny)
      END IF
      IF( char(i,j+1)  AND temp(i,j+1)=0) THEN
         id = FILL( UP , bin, i, j, char, temp, nx , ny)
      END IF
      IF( char(i,j-1)  AND temp(i,j-1)=0) THEN
         id = FILL( DOWN , bin, i, j, char, temp, nx , ny)
      END IF
   END IF
END PROCEDURE
```

Figure 6.14: The 2D AMR algorithm used to form rectangles around the flagged points

```
PROCEDURE FORM_RECT( nx,ny,temp,bins, imin, jmin, icount, jcount)
Given the array, temp , form rectangles
   set imin array = nx, jmin array = ny,
   icount array =-1, jcount array = -1
   DOj = 1 , ny
      DOi = 1 , nx
         IF( temp(i,j) ≠ 0 ) THEN
            IF( i ≤ imin(temp(i,j)) THEN
               imin(temp(i,j) = i
            END IF
            IF( j ≤ jmin(temp(i,j)) THEN
               jmin(temp(i,j) = j
            END IF
            IF( i ≥ icount(temp(i,j)) THEN
               icount(temp(i,j) = i
            END IF
            IF( j ≥ jcount(temp(i,j)) THEN
               jcount(temp(i,j) = j
            END IF
         END IF
      END DO
   END DO
   DOi = 1 , bins
      icount(i) = icount(i) - imin(i) + 1
      jcount(i) = jcount(i) - jmin(i) + 1
   END DO
   DOi = 1 , bins
      IF( MOD(icount(i),2) = 0 ) icount(i) = icount(i) + 1
      IF( MOD(jcount(i),2) = 0 ) jcount(i) = jcount(i) + 1
   END DO
END PROCEDURE

PROCEDURE RESIZE_RECT(( nx,ny,bins ,imin,jmin,icount,jcount)
Makes sure that the rectangles have 2^n + 1 points per side
   DOi = 1 , bins
      Add points to icount(i) , and change imin(i)
      Add points to jcount(i) , and change jmin(i)
   END DO
END PROCEDURE
```

Figure 6.15: The output of the first run in testing the 1D AMR routine

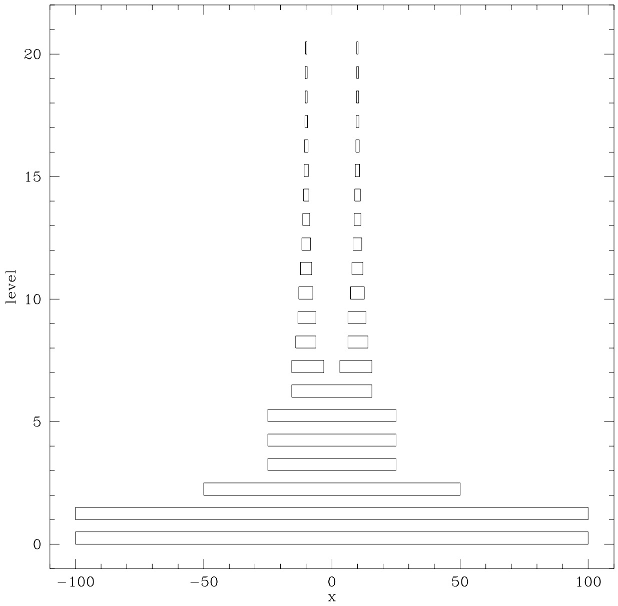
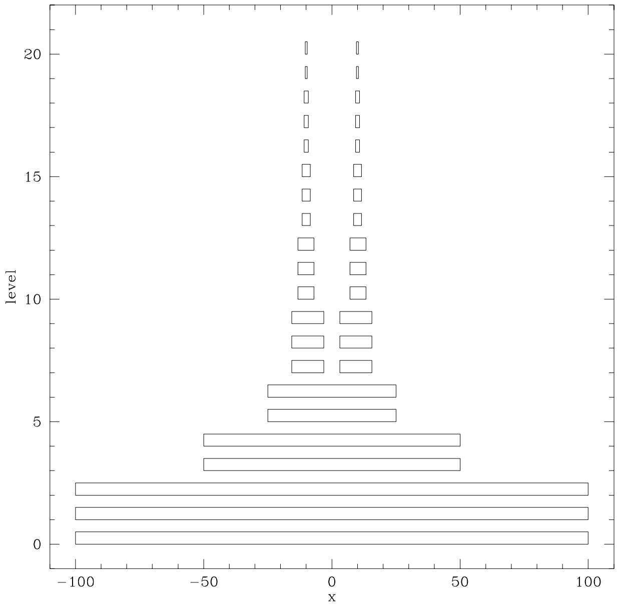


Figure 6.16: The output of the second run in testing the 1D AMR routine



1D AMR test, output for run3

l	GXMIN	GXMAX	NX
0	-1.00000e+02	1.00000e+02	3
1	-1.00000e+02	1.00000e+02	5
2	-5.00000e+01	5.00000e+01	5
3	-2.50000e+01	2.50000e+01	5
4	-2.50000e+01	2.50000e+01	9
5	-1.25000e+01	1.25000e+01	9
6	6.25000e+00	1.25000e+01	5
6	-1.25000e+01	-6.25000e+00	5
7	-1.25000e+01	-6.25000e+00	9
7	6.25000e+00	1.25000e+01	9
8	7.81250e+00	1.25000e+01	13
8	-1.25000e+01	-7.81250e+00	13
9	-1.17188e+01	-7.81250e+00	21
9	7.81250e+00	1.17188e+01	21
10	8.59375e+00	1.13281e+01	29
10	-1.13281e+01	-8.59375e+00	29

Figure 6.17: The output of the third run in testing the 1D AMR routine

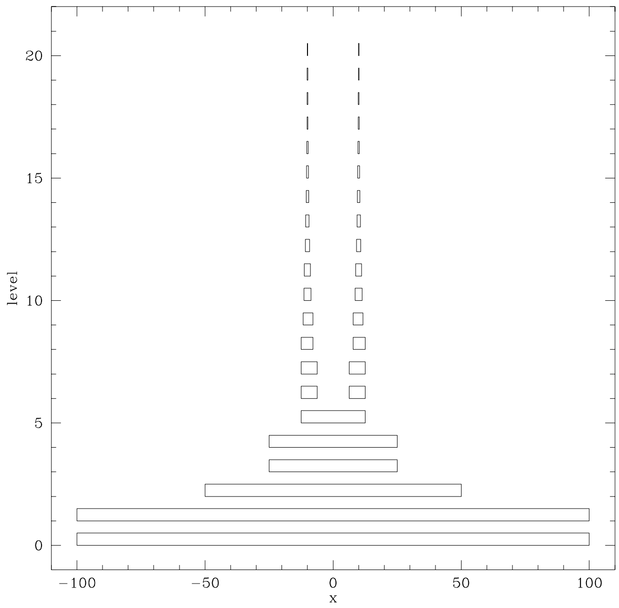


Table 6.8: 2D AMR test parameters

run number	option	α	β	γ	c_1	c_2	radius1	radius2
1	1	1.0	3.0	1.0	(0.0,-5.0)	(0.0,5.0)	3.0	1.0
2	0	1.0	3.0	1.0	(0.0,-5.0)	(0.0,5.0)	3.0	1.0
3	0	1.0	2.0	2.0	(0.0,-10.0)	(0.0,10.0)	0.5	0.5

Figure 6.18: The output of the first run in testing the 2D AMR routine

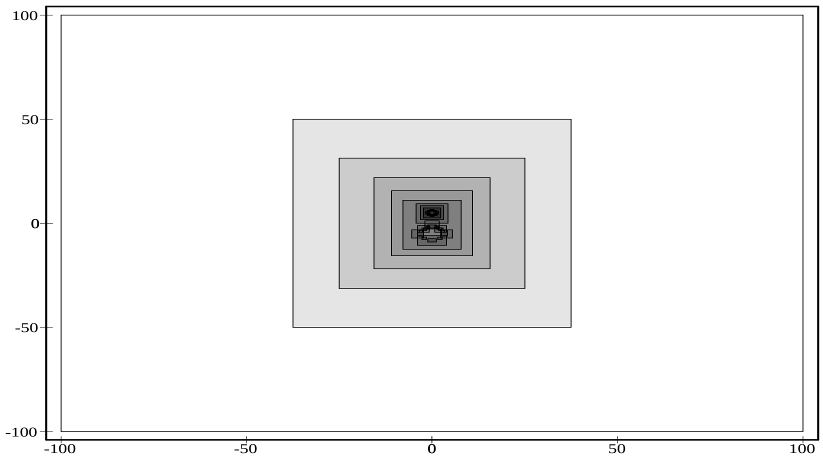


Figure 6.19: A close-up view of the first run in testing the 2D AMR routine

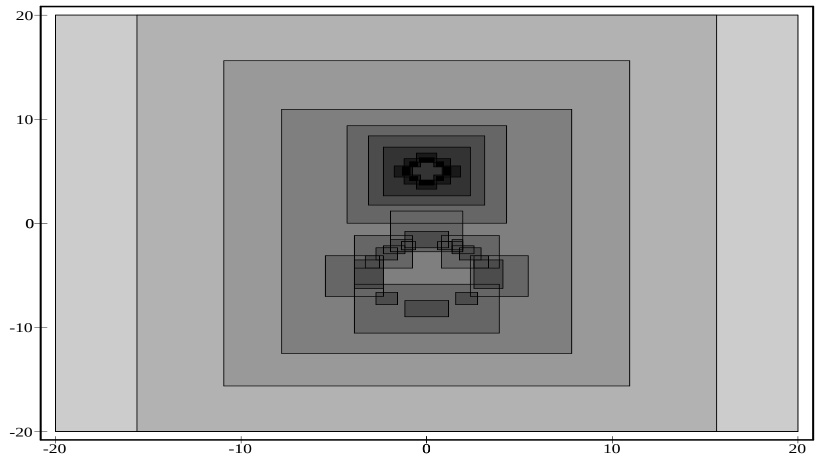


Table 6.9: 2D AMR test, output for run1

l	GXMIN	GXMAX	GYMIN	GYMAX	NX	NY	tau_avg
0	-1.000e+02	1.000e+02	-1.000e+02	1.000e+02	17	17	2.648e-04
1	-3.750e+01	3.750e+01	-5.000e+01	5.000e+01	13	17	4.371e-03
2	-2.500e+01	2.500e+01	-3.125e+01	3.125e+01	17	21	4.575e-03
3	-1.563e+01	1.563e+01	-2.188e+01	2.188e+01	21	29	5.482e-03
4	-1.094e+01	1.094e+01	-1.563e+01	1.563e+01	29	41	4.442e-03
5	-7.813e+00	7.813e+00	-1.250e+01	1.094e+01	41	61	1.686e-03
6	-4.297e+00	4.297e+00	0.000e+00	9.375e+00	45	49	1.558e-03
6	-1.953e+00	1.953e+00	-2.734e+00	1.172e+00	21	21	1.931e-04
6	7.813e-01	3.906e+00	-4.297e+00	-1.172e+00	17	17	1.782e-04
6	-3.906e+00	-7.813e-01	-4.297e+00	-1.172e+00	17	17	1.782e-04
6	2.344e+00	5.469e+00	-7.031e+00	-3.125e+00	17	21	2.023e-04
6	-5.469e+00	-2.344e+00	-7.031e+00	-3.125e+00	17	21	2.023e-04
6	-3.906e+00	3.906e+00	-1.055e+01	-5.859e+00	41	25	1.445e-04
7	1.563e+00	2.734e+00	-7.813e+00	-6.641e+00	13	13	1.083e-04
7	-2.734e+00	-1.563e+00	-7.813e+00	-6.641e+00	13	13	1.083e-04
7	-1.172e+00	1.172e+00	-8.984e+00	-7.422e+00	25	17	1.309e-04
7	-3.906e+00	-2.344e+00	-6.250e+00	-3.516e+00	17	29	1.180e-04
7	2.539e+00	4.102e+00	-6.250e+00	-3.516e+00	17	29	1.082e-04
7	-1.367e+00	-7.813e-01	-2.539e+00	-1.758e+00	7	9	8.211e-05
7	-2.344e+00	-1.172e+00	-2.930e+00	-2.148e+00	13	9	9.926e-05
7	-2.734e+00	-1.563e+00	-3.516e+00	-2.344e+00	13	13	1.157e-04
7	-3.320e+00	-2.539e+00	-4.297e+00	-3.125e+00	9	13	5.541e-05
7	7.813e-01	1.953e+00	-2.539e+00	-1.758e+00	13	9	5.348e-05
7	1.367e+00	2.539e+00	-2.930e+00	-2.148e+00	13	9	1.020e-04
7	1.758e+00	2.930e+00	-3.516e+00	-2.344e+00	13	13	1.016e-04
7	2.539e+00	3.320e+00	-4.297e+00	-3.125e+00	9	13	5.541e-05
7	-1.172e+00	1.172e+00	-2.344e+00	-7.813e-01	25	17	1.174e-04
7	5.859e-01	1.367e+00	-2.539e+00	-1.758e+00	9	9	1.041e-04
7	-1.367e+00	-5.859e-01	-2.539e+00	-1.758e+00	9	9	1.041e-04
7	1.367e+00	1.953e+00	-2.734e+00	-1.563e+00	7	13	4.155e-05
7	-1.953e+00	-7.813e-01	-2.734e+00	-1.563e+00	13	13	5.322e-05
7	-3.125e+00	3.125e+00	1.758e+00	8.398e+00	65	69	7.428e-04

Figure 6.20: A close-up view of the hole for run 1, used in testing the 2D AMR routine

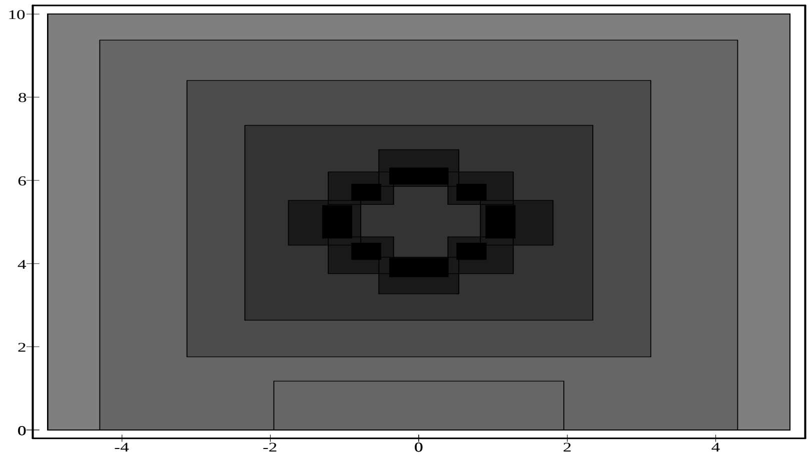


Figure 6.21: The output of the second run in testing the 2D AMR routine

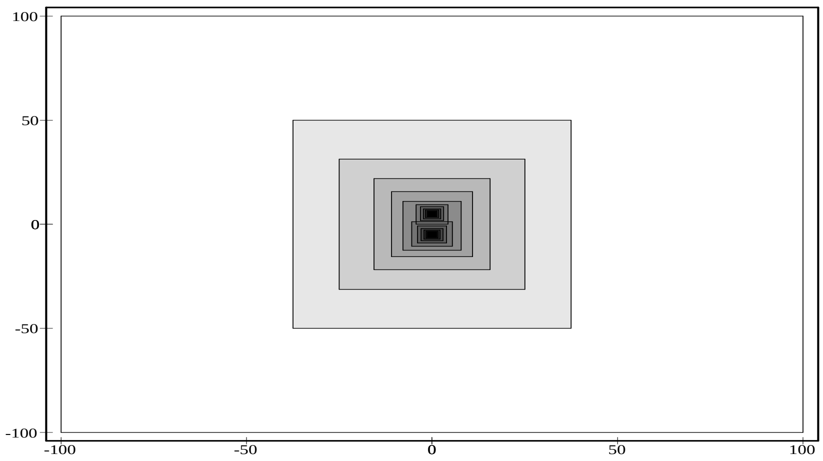


Figure 6.22: A close-up view of the second run in testing the 2D AMR routine

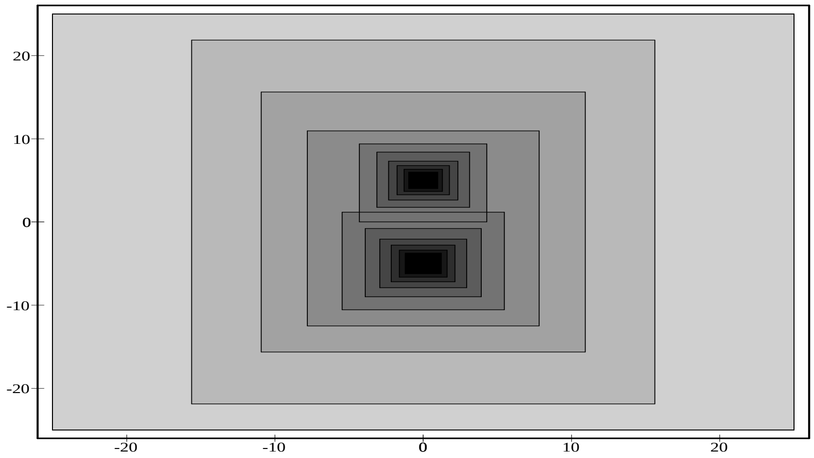


Figure 6.23: A close-up view of the hole for run 2 used in testing the 2D AMR routine

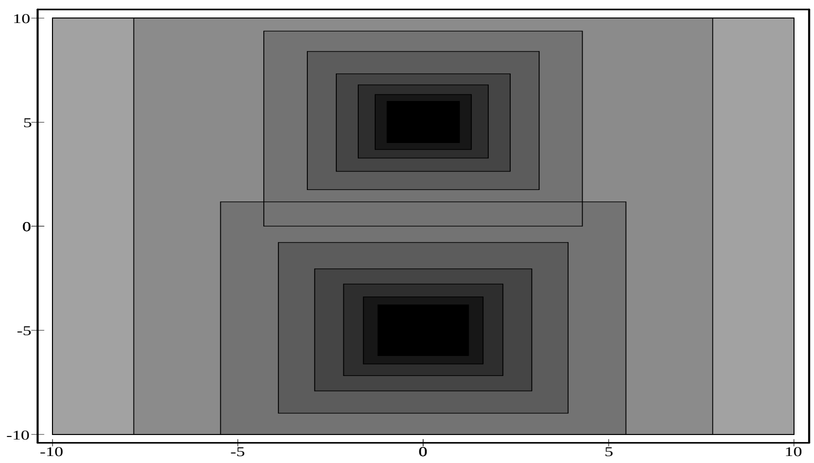


Figure 6.24: The output of the third run in testing the 2D AMR routine

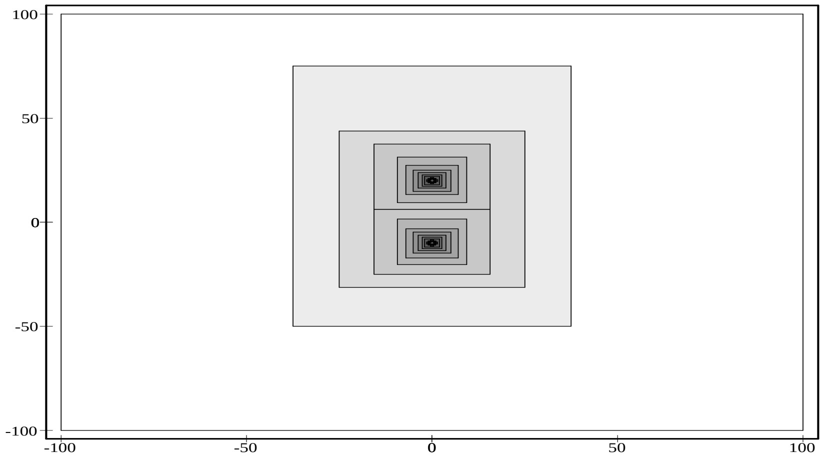


Figure 6.25: A close-up view of the third run in testing the 2D AMR routine

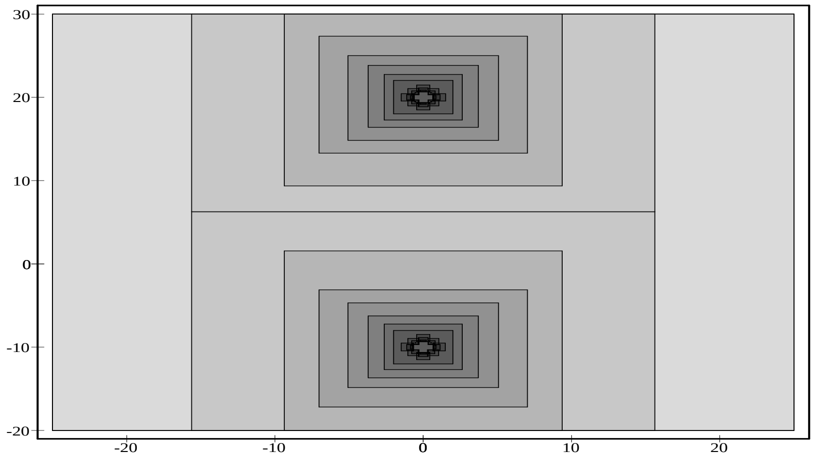
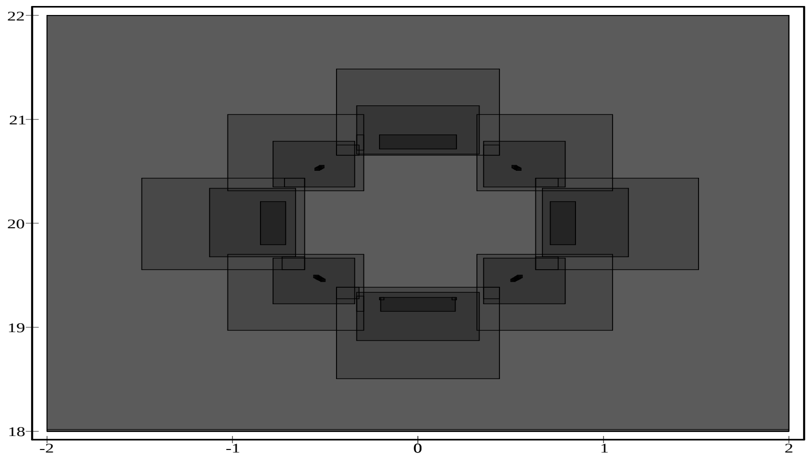


Figure 6.26: A close-up view of the hole for run 3 used in testing the 2D AMR routine



6.3 Examination of the outer boundary in 1D

In this section we examine the application of a Robin outer boundary condition in one dimension. The ideas that we will tie into this section are boundary smoothing, boundary transfers, deferred correction and Richardson extrapolation. The goals of this section are simple. We wish to develop techniques to incorporate the outer boundary condition,

$$u \to 1 \qquad \text{as} \qquad r \to \infty$$

with multigrid techniques. However we do not provide a mechanism to treat an unbounded domain with this code; instead, we will always impose the outer boundary condition at some fixed location, $x_{\max}$. We mentioned earlier that the boundaries were to be handled separately from the interior points, and in this section we describe the algorithms to handle this boundary condition along with our implementation of deferred correction .

Our model problem is now

$$\begin{aligned} \frac{\partial^2 u(x)}{\partial x^2} &= f(x) \quad x = 1 \cdots x_{\max} && (6.9) \\ \frac{\partial u}{\partial x} &= \frac{1-u}{x} \quad x = x_{\max} && (6.10) \\ u(1) &= u(x=1), \end{aligned}$$

where $x_{\max} = 3$. Again, we specify the solution

$$u(x) = 1 + \frac{1}{x},$$

which requires

$$f(x) = \frac{2}{x^3}.$$

These equations are differenced as

$$\frac{u[j+1] - 2u[j] + u[j-1]}{h^2} = f[j] \quad j = 2 \cdots n-1, \tag{6.11}$$

and

$$\frac{u^*[j] - u[j-1]}{h} - \frac{1 - u^*[j]}{x_{\max}} = g \quad j = n \tag{6.12}$$

where $x_{\max}$ is the position of the outer boundary, $u^*[j]$ is an intermediate value of $u[j]$, and g is the right hand side of the outer boundary equation, which is zero. These equations are represented as

$$\begin{aligned} L^h u^h[j] &= f^h[j] \quad j = 2 \cdots n-1 \\ Z_1^h u^h[j] &= g^h[j] \quad j = n. && (6.13) \end{aligned}$$

We only discretize the boundary equation to first order since we will make use of deferred correction to recover an expandable second-order solution. Now, the boundary operators we use do not contain the constant terms such as

$$\frac{1}{x_{\max}} \tag{6.14}$$

in equation (6.12). The fourth order boundary operator which we will use for deferred correction is

$$\begin{aligned} Z_4^h u^h[n] &= \frac{\frac{25}{12}u[n] - 4u[n-1] + 3u[n-2] - \frac{4}{3}u[n-3] + \frac{1}{4}u[n-4]}{h} \\ &+ \frac{u[n]}{x[n]}, \end{aligned} \tag{6.15}$$

and the second order boundary operator is defined by

$$Z_2^h u^h[n] = \frac{3u[n] - 4u[n-1] + u[n-2]}{2h} + \frac{u[n]}{x[n]}. \tag{6.16}$$

We use this second-order operator for test purposes only, in order to verify that the solution of the difference equations will not be extrapolatable when this operator is used. Finally, the first order operator is:

$$Z_1^h u^h[n] = \frac{u[n] - u[n-1]}{h} + \frac{u[n]}{x[n]}. \tag{6.17}$$

We now treat this boundary equation separately from the interior equations. Since we have a uniform domain, the boundary equation will be only partially relaxed using

$$u[j] := u[j] + \alpha u^*[j] \quad j = n, \tag{6.18}$$

where $\alpha \geq 0$ is the relaxation factor. The interior transfers use the full weighted restriction operator defined by equation (4.9), and our prolongation operator is given by equation (4.11). The transfers for the boundary points will all be injections, *i.e.*, the restrictions are defined by

$$u^H[N] = I_h^H u^h[n] = u^h[n] \tag{6.19}$$

where $n = 2N - 1$ and the prolongations are defined by

$$u^h[n] = I_H^h u^H[N] = u^H[N] \tag{6.20}$$

The relative truncation error estimate for the boundary point is determined from

$$\tau_h^H[N] = Z_1^H I_h^H u^h[n] - I_h^H Z_1^h u^h[n], \tag{6.21}$$

where

$$Z_1^h u^h[n] = \frac{u[j] - u[j-1]}{h} + \frac{u[j]}{x[j]} \quad j = n. \tag{6.22}$$

We will use deferred correction only on the finest level of the V-cycle. Thus, when we use a FMG routine, the deferred correction routine gets invoked directly after the finest level of each problem is to be smoothed. The order used in the deferred correction algorithm is a input parameter which allows us to examine the minimum order needed in ordr to get extrapolatable results. Since we saw in section(4.9) that boundary equations like this will not produce extrapolatable results when only differenced to second order, we should expect to see this non-extrapolatable behavior.

6.3.1 1D MG FAS algorithm using outer boundary condition

We will only describe the algorithm for the V-cycle (Figure 6.27) since for our problem a V-cycle always converges more rapidly than any other μ cycle. As we just stated, we see that the deferred correction is only invoked on the finest level, just before a coarse grid correction occurs.

Figure 6.27: The V-cycle FAS algorithm with incorporation of deferred correction for the outer boundary

PROCEDURE VCYCLE_FAS(ncycle, $l_{\max}$, pre, pst , order)
DOcycle= 1 ,ncycle
Cycle up to the coarsest level
DOl $= l_{\max}, l_{\text{coarse}} + 1$
IF(l $\neq l_{\max}$ OR cycle= 1) THEN
DOp = 1 , pre
RELAX_INTERIOR($u^h[j] = f^h[j]$) $\quad j = 2 \cdots n-1$
RELAX_BOUNDARY($u^h[n] = g^h[n], \alpha$) j=n
END DO
END IF
IF (l $= l_{\max}$) THEN
Now apply deferred correction
IF(ORDER = 2) THEN
$g^h[n] := Z_2^h(u[n]) - Z_1^h(u[n])$
Since $g^h = 0$ on the finest level, we determine g^h
only from the difference of the operators
ELSE IF(ORDER = 4) THEN
$g^h[n] \to Z_4^h(u[n]) - Z_1^h(u[n])$
END IF
END IF
$\tau_h^H[J] = L^H I_h^H u^h[j] - I_h^H L^h u^h[J], \qquad J = 2 \cdots n-1$
$\tau_h^H[N] = Z_1^H I_h^H u^h[n] - I_h^H Z_1^h u^h[n]$
$f^H[J] = I_h^H f^h[j] + \tau_h^H[J]$ *Correct the rhs*
$g^H[N] = I_h^H g^h[n] + \tau_h^H[n]$ *Correct the rhs of the outer boundary*
END DO
SOLVE($L^H u^H = f^H$, and $Z_1^H u^H = g^H$)
DOl $= l_{\text{coarse}}, l_{\max} - 1$
$u^h[j] := u^h[j] + I_H^h \left(u^H[J] - I_h^H u^h[j]\right) , \qquad j = 2 \cdots n-1$
$u^h[n] := u^h[j] + I_H^h \left(u^H[N] - I_h^H u^h[n]\right) ,$ j = n
Now smooth out u^h
DOq = 1 , pst
RELAX_INTERIOR($u^h[j], f^h[j]$) $\quad j = 2 \cdots n-1$
RELAX_BOUNDARY($u^h[n], g^h[n], \alpha$) j=n
END DO
END DO
END DO
END PROCEDURE

6.3.2 Results

The first table, Table 6.10, presents the results of an experimental study to determine which α gives the fastest convergence. This table shows the convergence rate versus α for an 8 level system. We have verified that the convergence rates, ρ, are the same for different level systems. We let `order` $= 1$ for these results so that deferred correction is disabled here. We also iterate until the residuals have been driven down to machine accuracy.[1] We clearly see that the best α is a small, but non-zero value.

In Table 6.11, we show the ℓ_1 norm of the error, e, of the difference solution relative to the exact solution (3.10) again computed with `order` $=1$. We expect first order convergence since the boundary condition is only treated to first order and this is what we observe. Here we set $\alpha = 1.0\mathrm{e}-10$ since we have seen that this will give the fastest convergence. Since we do not get second order results, there is no need to check for extrapolatable results. The order that we are getting is determined from

$$O = \log_2\left(\frac{e_{l+1}}{e_l}\right)$$

Thus, the order for this output is determined from

$$O = \log_2\left(\frac{2.18\mathrm{e}-05}{4.38\mathrm{e}-05}\right) \approx 1.0$$

Now it will be informative to see if we could use `order`$= 2$, and get extrapolatable results. We see that in Table 6.12, the errors clearly go down by a factor of four. We also see that the convergence rate slows down quite a bit. As we mentioned previously, in the asymptotic limit, the convergence rate will be that of the higher order operator, even though the algorithm relaxes the boundary point only to first order. Figure 6.28 shows a logarithmic plot of the errors. We only plot 17 points on each level, and the larger symbols correspond to the coarser levels. The dotted lines in the figure are evenly spaced lines which are factors of $\log_{10}(2)$ apart from each other. Thus, fourth order errors should be signaled by points which are separated by 4 lines. Clearly, the results show that the errors are not fourth order, but rather third order. Heuristically, we can argue that this occurs because the outer boundary was only corrected to second order, so that u apparently has an expansion:

$$u^h = u + h^2 e_2 + h^3 e_3 + O\left(h^4\right). \tag{6.23}$$

Thus, if we try to use Richardson extrapolation, we will not produce $O\left(h^4\right)$ results. Now we should be able to produce an extrapolatable solution using deferred correction of the

[1] We define ϵ (machine accuracy) such that ϵ is the smallest floating point number such that $1.0 + \epsilon = 1.0$.

Table 6.10: Results from the 1D FAS FMG algorithm with outer boundary condition.

examining α

α	$\rho(\alpha)$
0.0(+0)	-1.5(-1)
1.0(-12)	-4.9(-1)
1.0(-10)	-1.9(+0)
1.0(-8)	-1.6(+0)
1.0(-6)	-1.3(+0)
1.0(-4)	-9.5(-1)
1.0(-2)	-4.5(-1)
0.5(+0)	-4.5(-2)
1.0(+0)	-3.6(-2)

boundary equation. If we look at the problem where we are using some non-conforming mesh structure but with a priori knowledge of the exact value of u at whatever lattice point the boundary condition is applied, then clearly we expect the solution to be extrapolatable. Now it is also true in the limit that the order of the boundary approximation, q, goes to infinity we must also recover this case (since we would not be truncating the Taylor series expansion). Now if we reduce q from infinity, then for some q, q' we will lose the ability to extrapolate, and then $q' + 1$ is the minimum order we need.

We now look into the case when `order` $= 4$, which is shown in Table 6.13. We see that the convergence rate once again comes down slightly, where we compute the convergence rate from the residuals of the interior equations. We also see that the errors, although second order, are slightly larger than they were when we relaxed the boundary to second order, but Figure 6.29 clearly shows that the errors of each level are 4 lines apart, *i.e.* the solution is fourth order. Thus, we see that we are getting extrapolatable results.

We have seen in this section that by using deferred correction, we can easily take a first order code, and get fourth order results via deferred correction and Richardson extrapolation.

Table 6.11: Results from the 1D FMG FAS algorithm with the outer boundary condition imposed; where `order` = 1

l	$\rho\,(l)$	$e\,(l)$
1	-1.1(+0)	3.0(-2)
2	-1.4(+0)	1.1(-2)
3	-1.5(+0)	4.3(-3)
4	-1.8(+0)	1.8(-3)
5	-1.9(+0)	7.9(-4)
6	-1.9(+0)	3.7(-4)
7	-1.9(+0)	1.8(-4)
8	-1.9(+0)	8.8(-5)
9	-2.0(+0)	4.4(-5)
10	-2.0(+0)	2.2(-5)

Table 6.12: 1D FMG FAS algorithm with outer boundary condition, where `order` = 2 and α = 1.0e-09

l	$\rho\,(l)$	$e\,(l)$
1	-8.2(-1)	1.1(-2)
2	-1.1(+0)	4.0(-3)
3	-1.2(+0)	1.1(-3)
4	-1.3(+0)	3.0(-4)
5	-1.4(+0)	7.6(-5)
6	-1.4(+0)	1.9(-5)
7	-1.5(+0)	4.8(-6)
8	-1.5(+0)	1.2(-6)
9	-1.5(+0)	3.0(-7)
10	-1.5(+0)	7.5(-8)

Table 6.13: 1D FMG FAS algorithm with outer boundary condition, where `order` = 4 and α = 1.0e-09

l	$\rho\,(l)$	$e\,(l)$
1	-8.1(-1)	7.4(-3)
2	-1.0(+0)	5.0(-3)
3	-1.2(+0)	1.4(-3)
4	-1.3(+0)	3.6(-4)
5	-1.4(+0)	9.1(-5)
6	-1.4(+0)	2.3(-5)
7	-1.4(+0)	5.7(-6)
8	-1.5(+0)	1.4(-6)
9	-1.5(+0)	3.6(-7)
10	-1.5(+0)	9.0(-8)

Figure 6.28: Errors in the extrapolated results with `order = 2`

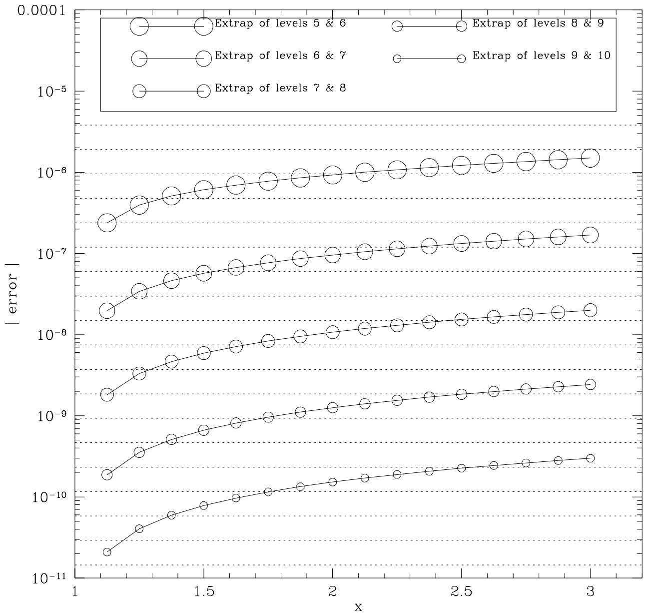
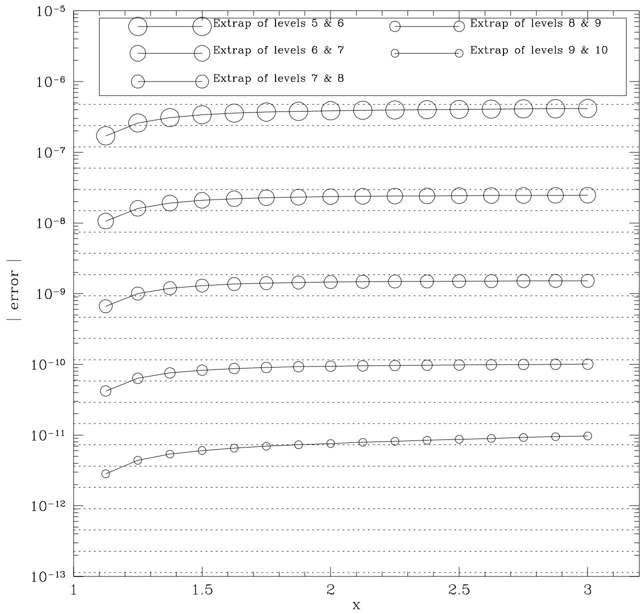


Figure 6.29: Errors in the extrapolatated results with `order = 4`



6.4 Extrapolation with MLAT

In this section we develop techniques which allow us to generate extrapolatable solutions from finite difference MLAT codes. One of the major reasons to use an adaptive grid is to generate an "unbounded" domain, (see for example Section 4.10). Since some of the grids in an adaptive grid/mesh structure do not cover the entire domain, we will see that "standard" MLAT algorithms do not produce extrapolatable results, and in fact, the solutions will generally only be first order even for second order differencing. The methods used to obtain extrapolatable results are certainly not well documented. This does not mean that one can not find extrapolatable solutions, it is just that people have not developed the methods.

We will start by defining a 1D model problem and then describe several attempts to obtain extrapolatable solutions. For must of the attempts, we only look at the case where there is one interface region. We believe that if we are able to generate an extrapolatable solution which has one interface region, then we should be able to use multiple interface regions. Next we compare these attempts and see which one yields an extrapolatable solution. Finally, we will use the techniques to generate an extrapolatable solution for a simple 2D problem.

6.4.1 1D model problem

The problem which we examine in this section is

$$\begin{aligned} \frac{\partial^2 u(x)}{\partial x^2} &= f(x) \qquad a < x < x_{\max} \\ u(a) &= u(x=a) \\ u(x_{\max}) &= u(x = x_{\max}) \end{aligned} \tag{6.24}$$

We finite-difference the interior equation to $O\left(h^2\right)$ to obtain

$$\begin{aligned} \frac{u[j+1] - 2u[j] + u[j-1]}{h^2} &= f[j], \qquad j = 2, \cdots, N-1 \\ u[1] &= u(x=a) \\ u[N] &= u(x = x_{\max}) \end{aligned} \tag{6.25}$$

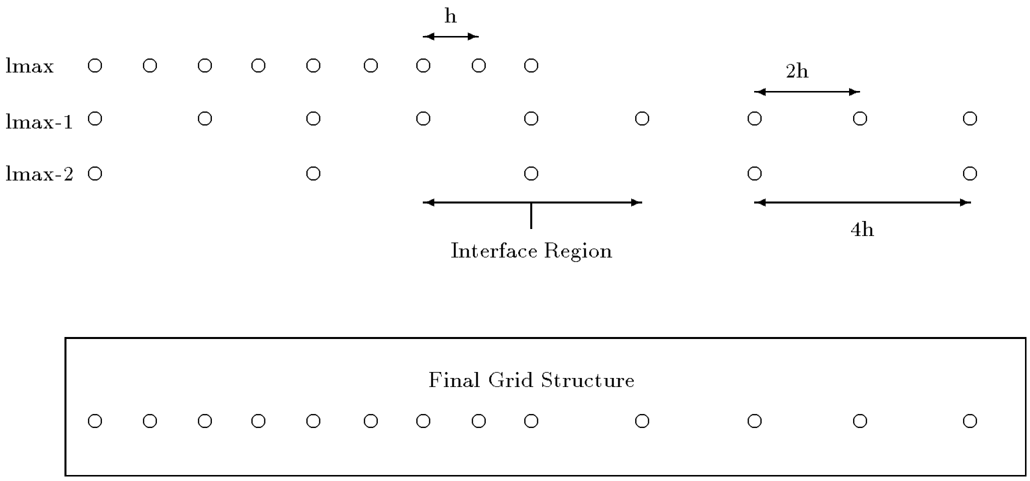
Once again we use

$$f(x) = \frac{2a}{x^3}. \tag{6.26}$$

so that the exact solution is

$$u(x) = 1 + \frac{a}{x}. \tag{6.27}$$

Figure 6.30: 1 interface region depicted in a 3 level scheme.



6.4.2 The 1D MLAT code

This code uses a FAS scheme which can be operated in a fully adaptive mode. For this code we will define l_c as the number of grids which have $x[n] = x_{\max}$ However, since we are trying to test whether we can obtain an extrapolatable solution, we have the user supply the grid structure before the code starts execution rather than let a structure be determined via an adaptive process. Specifically, we input a slight generalization of the the grid structure previously determined in Section 6.2.1. We do this because we are only trying to test whether the solutions to the equations differenced on an adaptive grid structure are extrapolatable. We have levels $l = 0 \cdots l_{\max}$, and when $l = l'$ a transition in behavior occurs. In this case the fine grid does not extend to as large a value as does the coarse grid:

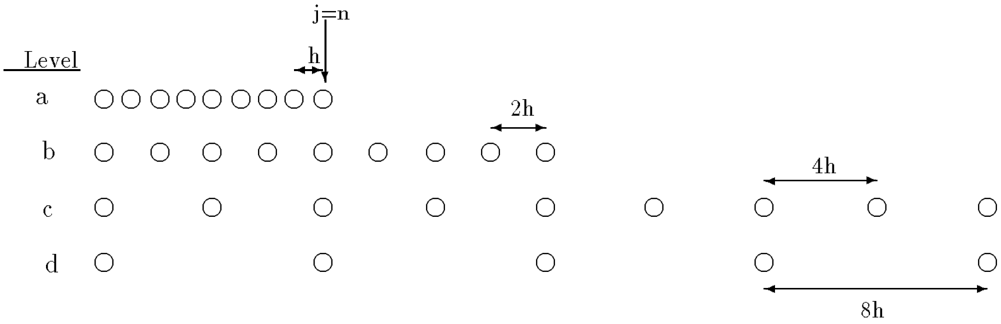
$$x^{h_{l'}}[n] \neq x^{H_l}[N]. \tag{6.28}$$

where N is the upper limit of the coarse grid and n is the upper limit of the fine grid. In this example we use

$$x^{h_{l'}}[n] = \frac{1}{2} x^{H_l}[N]. \tag{6.29}$$

If we generalize this to a number of levels, the number of points in each grid is the same. Figure 6.30 shows an example of this grid structure, where the transition occurs at $l = l_{\max} - 1$. In the algorithm we will use for this grid structure, we will take `pre` $= 2$ and `pst` $= 1$. We will also allow for the case when the boundary equation at $x = x_{\max}$ will be either a Dirichlet condition, or the Robin condition defined by equation (6.10).

Figure 6.31: The adaptive grid structure.



The major difference between this algorithm and the ones described in the previous sections is that $\tau_h^H[J]$ is not defined at every point. We see in Figure 6.30 that we have 3 levels. The grid at the finest level does not cover the entire domain, thus, we choose the convention that

$$\tau_h^H[J] \equiv 0, \tag{6.30}$$

where the points on the finest grid are not defined. This brings about a major problem, $\tau_h^H[J]$ is not continuous, and thus, not smooth. An example of this is shown in Figure 6.37 where the interface point is at $J = 17$. We have tried an enormous amount of unsuccessful methods, and will only describe a few of them, with the last description being of the successful method.

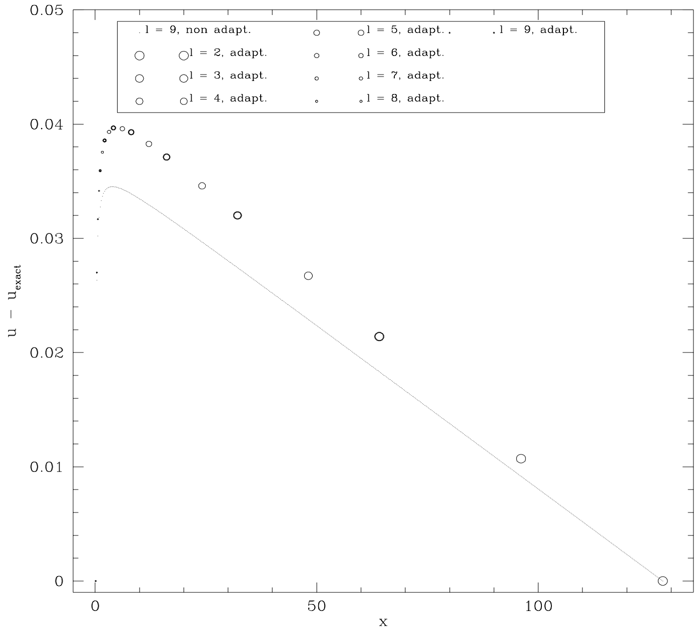
In this subsection we describe the "usual" algorithm[12] used when working with adaptive grids such as the one in Figure 6.31. This algorithm is based on the assumption that each parent has only one child; this is the case for the test problems in this chapter. The interface boundaries are treated as Dirichlet conditions here, thus $\tau_h^H[J] = 0$ at the interface. [2]

Figure 6.31 shows an example of the computational domain used by the algorithm. The algorithm starts by perform `pre` smoothing sweeps on grid (a). The value of $u[j]$ at $j = n$ is a Dirichlet condition, and thus, we do not change the value when we smooth. To determine this value at $j = n$ we use an interpolation from the points on grid (b). The interpolation routine that we use is injection from the corresponding coarse grid values from grid (b).

We first test the algorithm to ensure the convergence rate, ρ, remains constant as we increase the number of levels. Here we use a grid system with 5 levels, with each of the

[2] Since Dirichlet conditions are exact conditions (*i.e.* not differenced conditions) their relative truncation error estimate is zero)

Figure 6.32: The errors using the standard 1D MLAT algorithm, with and without adaptive grids.



three finest grids spanning half of the domain of their parents. We let $x_{\max} = 4$, $a = 1.0$ and let `outer_boundary = FALSE` for this system. The work unit is defined to increase by one every time we relax over s points, where s is the total number of points on the final finest grid. We see that in Figure 6.30, that $s = 9 + 4 = 13$, since there are 13 points on the final finest grid. Table 6.14 shows that the convergence rate remains approximately constant as the levels get finer. The convergence rate shows that for about every 2 work units, the residuals are reduced by 6 orders of magnitude. Figure 6.32 shows a graph of the error on the overall grid, e versus x. In this graph we see the errors from the adaptive mesh, as well as the errors from a non-adaptive mesh. We see that when we compare the adaptive run versus the non-adaptive run, the total number of points was tremendously decreased, while the error remained virtually unchanged.

The goal of adaptive mesh refinement is to minimize the total computational work which generally means minimizing the number of points used on the finest grid while ensuring that the overall level of error is comparable to that generated by a run where the finest level of discretization is used everywhere. Table 6.15 shows the ℓ_1 norm of the errors on a non-adaptive grid structure as a function of the finest level of discretization used. In this case the coarsest grid contains 3 points. $Q\,(l, l+1)$ is the reduction in error from level l to level $l+1$. We see that after 4 levels the reduction factor is 4.0 indicating second-order convergence. Table 6.16 also shows the reduction of error from one level to the next, this time for an adaptive run. Here we see that the ℓ_1 norm of the errors are very close in comparison to the non-adaptive run. Our problem though, is that these results are not second order accurate. They are *first order*, which makes them impossible to extrapolate. We also know that as we increase the number of levels, then these results will deviate much further from each other, since one result is first order, and the other is second order.

The adaptive code is not second order because it is not handling the boundaries at the interfaces correctly. We will see later, that if we do handle the boundaries correctly, then we will obtain a second order extrapolatable solution.

Finally we estimate the work units used by the algorithm. Here we will only estimate this for V-cycles. For the grid structures that we use, we define

$$\nu = \mathrm{Total number of grids where} x[n] = x_{\max}. \tag{6.31}$$

The total amount of work for the refinements we use in this section is

$$\begin{aligned} w_A &= \sigma\left(\mathtt{P}\ \nu\frac{\mathtt{nf}}{n} + \frac{\frac{1}{2}(\mathtt{nf}+1)}{n} + \frac{\frac{1}{4}(\mathtt{nf}+1)}{n} + \frac{\frac{1}{8}(\mathtt{nf}+1)}{n} + \cdots\right) \\ &\approx \sigma\mathtt{P}\,(\nu+1)\,\frac{\mathtt{nf}}{n} \end{aligned} \tag{6.32}$$

where $\mathtt{P}$ is the total number of smoothing sweeps per grid, for each cycle, $\mathtt{nf}$ is the number of grid points on the finest grid , σ is the total number of V-cycles and n is the total number of grid points for the fine grid equations. In general $n = (\mathtt{nf}-1)/2(\nu-1)$.

Figure 6.33: The V-cycle for the FAS MLAT algorithm

```
PROCEDURE VCYCLE_FAS_adaptive( ncycle, l_max, pre,
                  pst , order,outer_boundary)
   DOcycle= 1 ,ncycle
       Cycle up to the coarsest level
       DOl = l_max, l_coarse + 1
          IF(l ≠ l_max OR cycle= 1 ) THEN
             FIND_INTERFACE(j_max)
             j_max = FIND_INTERFACE will find the interface point
             on the coarse grid
             DOp = 1 , pre
                RELAX_INTERIOR(u^h(j) = f^h(j))     j = 2 ··· n − 1
                Smooth the residuals
                IF( outer_boundary =  TRUE  AND gmax(lhead(l))=x_max) THEN
                   RELAX_BOUNDARY(u^h(n) = g^h(n),α)     j=n
                END IF
                Smooth the boundary
             END DO
          END IF
          IF ( GMAX(lhead(l))= x_max  AND
                   gmax(gpch(lhead(l))) ≠ x_max ) THEN
             Now apply deferred correction to the outer boundary
          END IF
          τ_h^H(J) = L^H I_h^H u^h(j) − I_h^H L^h u^h(j),     J = 2 ··· J_max-1
          determine the truncation error estimate
          for the interior points
          τ_h^H(j) = 0  ,   j =j_max··· n
          set τ_h^H(j) to zero where it is not defined
          IF( outer_boundary =  TRUE  AND
                      gmax(lhead(l))=x_max) THEN
             determine the truncation error estimate for the outer boundary point
          END IF
          f^H(j) = I_h^H f^h(j) + τ_h^H(j)     Correct the rhs
          NOTE: when j ≥ j_max then determine f^H(j) from the analytical rhs
       END DO
```

Figure 6.33 continued

```
        Now solve the system on the coarsest level
         IF( outer_boundary =  TRUE) THEN
             SOLVE(L^H u^H = f^H,  and  Z_1^H u^H = g^H)
        ELSE
            SOLVE(L^H u^H = f^H)
        END IF
        Now come back down to the finest level
        DOl = l_coarse, l_max - 1
             Determine the correction
            u^h(j) := u^h(j) + I_H^h (u^H(J) - I_h^H u^h(j))  ,     j = 2 ··· j_max-1
            IF( outer_boundary =  TRUE  AND
                           gmax(lhead(l))=x_max) THEN
                u^h(n) := u^h(n) + I_H^h (u^H(N) - I_h^H u^h(n))  ,     j = n
            END IF
             Now smooth out u^h
            DOq = 1  , pst
                RELAX_INTERIOR(u^h(j), f^h(j))     j = 2 ··· n - 1
                Smooth the residuals
                IF( outer_boundary=  TRUE AND gmax(lhead(l))=x_max) THEN
                    RELAX_BOUNDARY(u^h(n), g^h(n), α)     j=n
                    Smooth the boundary
                END IF
             END DO
         END DO
     END DO
END PROCEDURE
```

6.4.3 The 1D hierarchal MLAT code

We will now look into another method since a "typical" MLAT algorithm does not give second order results when we let the grids adapt. One method which does work is given in pseudo-code form in figures (6.34) and (6.35). We see that in this code the algorithm cycles independently on each grid. The only inter-grid communication occurs at internal grid interface points. We transfer values at these interface point via injection from parental grids. Thereafter, the algorithm performs V-cycles using Dirichlet conditions everywhere. To summarize, this algorithm will use the FMG scheme in which each fine grid problem will have a Dirichlet condition imposed at the interface point. Thus, when the algorithm performs the V-cycles, there are no interfaces.

We see that this algorithm should take more work units than the MLAT algorithm in the last section. In fact the total work for σ v-cycles is

$$\begin{aligned} w_B &= \mathtt{P}\sigma\nu\left(\frac{\mathtt{nf}}{n} + \frac{\frac{1}{2}(\mathtt{nf}+1)}{n} + \frac{\frac{1}{4}(\mathtt{nf}+1)}{n} + \frac{\frac{1}{8}(\mathtt{nf}+1)}{n} + \cdots\right) \\ &\approx 2\,\mathtt{P}\,\sigma\,\nu\frac{\mathtt{nf}}{n} \end{aligned} \tag{6.33}$$

Thus comparing equations (6.32) and (6.33), we see that this algorithm takes about twice as much work as the standard MLAT method. However, we will now verify that this hierarchical technique is second order.

Table 6.17 shows the ℓ_1 norm of the errors for the hierarchal MG code. Here we see that the results from this routine are extremely inaccurate compared to those from the standard MLAT code. Using 13 levels, the error is only reduced to that of a 4 level run of the last algorithm. In order for the error of this routine to be smaller than the MLAT code, we would need to have 25 levels of refinement, which is not a realistic number of levels. We will now look back at the MLAT algorithm, and attempt to modify it to obtain extrapolatable results.

Figure 6.34: The hierarchal FMG FAS routine.

PROCEDURE FMG_Hierarchal(cycle, $l_{\max}$, pre, pst , order)
    Initialize Variables
    Initialize Memory
    DOl $= 0, l_{\max}$
        IF(l $\neq 0$) THEN INTERPOLATE(u^H)
        This interpolation will inject the interface point into the parent grid
        VCycle_H( )
    save the solution at the interface point
    COPY (interface(lhead(l)) = $u^H(J_{\max})$))
    END DO *for l*
END PROCEDURE

Figure 6.35: The hierarchal FMG FAS V-cycle routine.

PROCEDURE VCYCLE_H(ncycle, $l_{\max}$, pre, pst , order)
    All grids will span the domain of the grid at $l_{\max}$.
    Temporary storage will be issued for all lower level grids.
    Initialize Variables
    Initialize Memory
    The coarsest level will always have 3 points
    DOcycle= 1 ,ncycle
        Cycle up to the coarsest level
        DOl $= l_{\max}, l_{\text{coarse}} + 1$
            IF(l $\neq l_{\max}$ OR cycle= 1) THEN
                DOp = 1 , pre
                    Smooth the residuals
                        Smooth the boundary
                        END DO
            END IF
            IF (GMAX(lhead(l))= $x_{\max}$ AND
                gmax(gpch(lhead(l))) $\neq x_{\max}$) THEN
                    Now apply deferred correction
                END IF
                $\tau_h^H(J) = L^H I_h^H u^h(j) - I_h^H L^h u^h(j)$, $\quad j = 2 \cdots$ n-1
                determine the truncation error estimate for the interior points
                IF(outer_boundary = TRUE AND
                                gmax(lhead(l))=$x_{\max}$) THEN
                    determine the truncation error estimate for the outer boundary point
                Correct the rhs of the outer boundar
            END IF
            $f^H(J) = I_h^H f^h(j) + \tau_h^H(J)$ *Correct the rhs*
        END DO

Figure(6.35) continued

```
        Now solve the system on the coarsest level
         IF( outer_boundary =  TRUE) THEN
            SOLVE(L^H u^H = f^H,  and  B_1^H u^H = g^H)
        ELSE
           SOLVE(L^H u^H = f^H)
        END IF
        Now come back down to the finest level, performing the needed corrections
        DOl = l_coarse, l_max - 1
           Determine the correction
           u^h(j) := u^h(j) + I_H^h (u^H(J) - I_h^H u^h(j))  ,     j = 2 ... n-1
           IF( outer_boundary =  TRUE) THEN
              u^h(n) := u^h(n) + I_H^h (u^H(N) - I_h^H u^h(n))  ,     j = n
           END IF
           Now smooth out u^h
           DOq = 1 , pst
              Smooth the residuals
              Smooth the boundary
           END DO
         END DO
     END DO
    Release all the temporary storage
END PROCEDURE
```

6.4.4 Making an adaptive MLAT code extrapolatable

To simplify matters, we will again work with a single-interface system in the following. To test the extrapolations we let the coarse grids, $L = 0, \cdots, L-1$ span the whole domain, and then let the fine grid, l span half of the domain. We position the grid interface near $x = a$ since the truncation error estimates will be large there. The goal is to get an error with the single-interface, adaptive run which is very close to the error achieved by the non-adaptive run, Table 6.15.

As mentioned previously, we can resolve our interface problems using a single interface then presumably the techniques will immediately extend to the case of multiple interfaces. We also emphasize that we want techniques that can be extended to multiple dimensions with little or no modifications. We will discuss three attempts in this section—only the third is successful.

The first attempt tries to use another boundary condition at the interface point of the finest system. Previously we were using a Dirichlet condition on the interface point, and the hope was that by using a Neumann condition, the solutions would become extrapolatable. In hindsight, we see that this could never work since it does not address the issue of the non-smoothness of τ_h^H.

The first attempt is based on a requirement of continuity of $\partial u/\partial x$ across the interface:

$$\frac{\partial u^h(x)}{\partial x} = \frac{\partial u^H(x)}{\partial x} \quad x = \frac{x_{\max}}{2} \tag{6.34}$$

which is finite differenced to fourth order as

$$\begin{aligned}(12h)^{-1} \Big(&25u^h[j] - 48u^h[j-1] + 36u^h[j-2] \\ &-16u^h[j-3] + 3u^h[j-4]\Big) = \\ &-12(H)^{-1} \Big(25u^H[J] - 48u^H[J+1] + 36u^H[J+2] \\ &-16u^H[J+3] + 3u^H[J+4]\Big),\end{aligned} \tag{6.35}$$

where again $j = 2J - 1$. This equation is only used when a coarse grid correction from the finest grid takes place. Since this is a one dimensional problem, and we do not care about the convergence rate at this time, we do not relax the boundary point on the finest level. Table 6.18 shows that the solution is once again only first order. We see that there is no difference between this solution and the original MLAT code. We also tried matching the second derivative at the interace, and once again found that these runs did not produce extrapolatable results.

The next attempt that we tried [42], was to use the coarse grid points around the interface point to introduce an (additional point) at $j = n + 1$. Figure 6.36 shows the grid structure for this attempt. We see that the point at $j = n$ on the finest grid can now be used as an interior point. Thus, all smoothing and transfers at this point are implemented as such. The only issue that remains is how to fix the function value at the point $j = n + 1$. From the results of Chapter 3, we know that if we use linear interpolation:

$$u^h[n+1] = \frac{1}{2}\left(u^H[J_{\max}] + u^H[J_{\max}+1]\right) + O\left(H^2\right) \tag{6.36}$$

where

$$x[J_{\max}] = x[n], \tag{6.37}$$

will will not get second order results. Thus, with the benefit of hindsight, we interpolate to fourth order as follows:

$$\begin{aligned} u^h[n+1] &= \frac{1}{16}\left(-u^H[J_{\max}+2] - u^H[J_{\max}-1]\right) \\ &+ \frac{9}{16}\left(u^H[J_{\max}] + u^H[J_{\max}+1]\right) \end{aligned} \tag{6.38}$$

Thus our full set of equations is:

$$\begin{aligned} h^{-2}\left(u^h[j+1] + u^h[j-1] - 2u^h[j]\right) &= f[j], \qquad j = 2, \cdots, n-1 \\ h^{-2}\left(u^h[j+1] + u^h[j-1] - 2u^h[j]\right) &= h^2 f[j], \qquad j = n \\ u^h[1] &= u\,(x = a) \end{aligned}$$

and

$$\begin{aligned} u^h[n+1] &= -\frac{1}{16}\left(u^H[J_{\max}+2] + u^H[J_{\max}-1]\right) \\ &+ \frac{9}{16}\left(u^H[J_{\max}] + u^H[J_{\max}+1]\right) \end{aligned}$$

on the fine grid, and

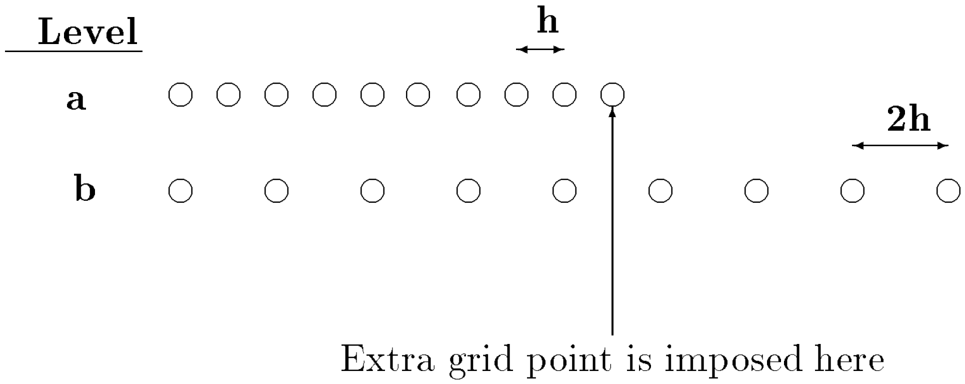
$$\begin{aligned} & H^{-2}\left(u^H[J+1] + u^H[J-1] - 2u^H[J]\right) = \\ & \quad I_h^H f[J] + \tau_h^H[J], \qquad J = 2, \cdots, N-1 \\ & \quad u^H[1] = u\,(a) \\ & \quad u^H[N] = u\,(x_{\max}) \end{aligned}$$

$$\tau_h^H[J] = L^H I_h^H u^h[j] - I_h^H L^h u^h[j], \quad J = 2, \cdots, J_{\max} \tag{6.39}$$

on the coarse grid, where

$$Lu[j] = \frac{u[j+1] + u[j-1] - 2u[j]}{h^2}$$

Figure 6.36: The 1D grid structure with a single interface.



and

$$I_h^H u^h[j] = \frac{1}{2}u[j] + \frac{1}{4}\left(u[j-1] + u[j+1]\right) \quad j = 2, \cdots, N$$

At this point, we will now define a new symbol that will be used in the tables. We will now signify the extrapolation between levels a and b as xab. Table 6.19 shows very strange behavior for the errors. We see that there is a region where the results appear to be extrapolatable. After the sixth level we do not show the extrapolation results, since $Q\,(l, l+1)$ deviates from 4.

We now look at $\tau_h^H[j]$ before we show the final algorithm. Figure 6.37 shows $\tau_h^H[j]$ on the points around the interface. We clearly see that the truncation error is non-smooth particularly at $J = J_{\max} - 1$. Thus, by trying to come up with a boundary condition for the point at $J = J_{\max}$, we are not able to help smooth $\tau_h^H[J_{\max} - 1]$.

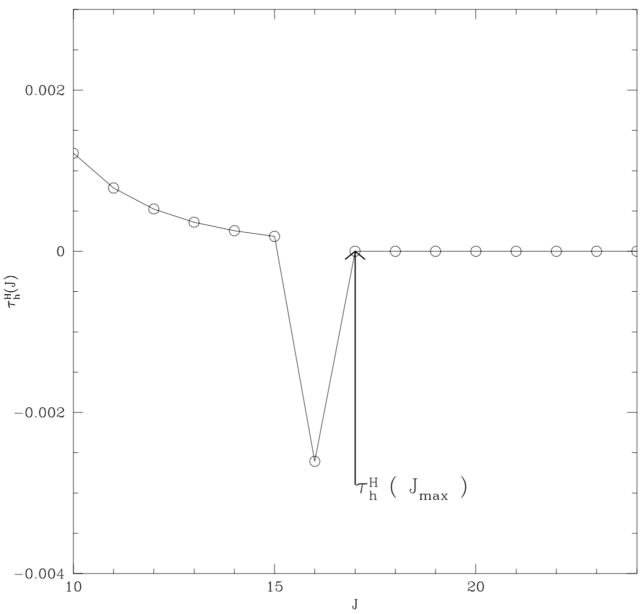
Our final algorithm takes the original MLAT algorithm and modifies it only in the manner that the truncation error estimate is calculated. Specifically, after computing τ_h^H in the usual fashion, we set

$$\tau_h^H[J_{\max} - 1] = \frac{1}{2}\left(\tau_h^H[J_{\max} - 2] + \tau_h^H[J_{\max}]\right). \tag{6.40}$$

By modifying the algorithm in this way we will not be able to look at the residuals and determine when the system has converged. We are now solving a different set of equations than that of equation (6.25). We have not yet come up with a specific scheme to determine convergence although Choptuik[14] points out that we could stop by testing for convergence, *i.e.* by looking for consistency in the extrapolation.

The output of these runs is shown in Table 6.20, with $a = 0.1$ so that, around the interface region, τ_h^H will be larger than the previous tests. We clearly see that the results are extrapolatable !

Figure 6.37: A zoom up on the relative truncation error



We now examine the case where we have multiple interfaces. For this run we let $a = 0.1, x_{\max} = 512$ and have 10 interfaces. We then vary the number of coarse grids, ν . Table 6.21 shows clearly that the solution is extrapolatable for the multi-interface case.

6.4.5 Two dimensional model problem

We now turn to a two dimensional problem, to see if we can smooth τ_h^H around the interface to get an extrapolatable answer. We will use the extremely simple grid structure, shown in Figure 6.38. We see that the grid interface is located along a constant-x line. This is a good test since the grid refinements that will be required of our fully adaptive MLAT code will be rectangles, such that each side of the rectangle will interface with the coarser grid along a line of coarse grid points.

Our 2 dimensional model problem is

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = f(x,y) \quad a < x < x_{\max} \quad -1 < y < 1 \tag{6.41}$$

where we use Dirichlet conditions around all of the boundaries and

$$u(x,y) = 1 + \frac{a}{\sqrt{x^2+y^2}}$$

Table 6.14: Convergence Rate in MLAT Dirichlet code using a 5 grid system.

l	$\rho(l)$
2	-1.6(+0)
3	-2.0(+0)
4	-2.4(+0)
5	-2.7(+0)
6	-2.8(+0)
7	-3.0(+0)
8	-3.0(+0)
9	-3.1(+0)
10	-3.1(+0)

$$f(x,y) = \frac{a}{(x^2+y^2)^{\frac{3}{2}}}. \tag{6.42}$$

In Figure 6.38 we define two separate regions, a smoothing region and a buffer region. In the buffer zone we set

$$\tau_h^H(x,y) = \frac{1}{4}\tau_H^{2h}. \tag{6.43}$$

Since τ_H^{2H} is smooth, and we are using a second order difference scheme, one quarter of this value is approximately τ_h^H. Secondly, since interpolation from one grid to another introduces high frequency components, we then smooth τ_h^H with the smoothing operator

$$\begin{aligned} \tau_h^H[j,k] := \frac{1}{4}\tau_h^H[j,k] &+ \frac{1}{8}\Big(\tau_h^H[j+1,k] + \tau_h^H[j-1,k] \\ &+ \tau_h^H[j,k+1] + \tau_h^H[j,k-1]\Big) \\ &+ \frac{1}{16}\Big(\tau_h^H[j+1,k+1] + \tau_h^H[j-1,k+1] \\ &+ \tau_h^H[j+1,k-1] + \tau_h^H[j-1,k-1]\Big) \end{aligned} \tag{6.44}$$

Table 6.22 shows that the solutions to this technique are extrapolatable. For this run we used $a = 0.125$, $x = a, \cdots, 2+a$,$y = -1, \cdots, 1$, with a buffer region of 4 zones, and a smoothing region of 8 zones.

We have now found a technique to generate extrapolatable solutions using an adaptive MLAT algorithm. This technique is by no means unique, although it works very well. We feel that it is likely that any technique that smooths τ_h^H along the interfaces will produce extrapolatable solutions. In the next section, we look into the issue of where the fine grids are not properly contained in parental grids.

Table 6.15: Convergence study of MLAT code using a 2 grid system, $x_{\max} = 4$

All grids span the entire domain

l	$e\,(l)$	$Q\,(l, l+1)$
1	2.2(-2)	2.7
2	6.5(-3)	3.4
3	1.7(-3)	3.8
4	4.3(-4)	3.9
5	1.1(-4)	4.0
6	2.7(-5)	4.0
7	6.8(-6)	4.0
8	1.7(-6)	4.0

Table 6.16: Convergence study of MLAT code using a 2 grid system, $x_{\max} = 4$

Adaptive run with Dirichlet boundary conditions imposed on all boundaries

l	$e\,(l)$	$Q\,(l, l+1)$
1	2.7(-2)	2.8
2	7.9(-3)	3.4
3	2.2(-3)	3.7
4	5.9(-4)	3.7
5	1.7(-4)	3.5
6	5.4(-5)	3.2
7	1.9(-5)	2.8
8	7.6(-6)	2.5

Table 6.17: Convergence study of hierachical MG code using a 2 grid system, $x_{\max} = 4$

Dirichlet boundary conditions imposed on all boundaries

l	$e\,(l)$	$Q\,(l, l+1)$
6	3.5(-1)	1.3
7	2.7(-1)	1.6
8	1.7(-1)	2.0
9	8.4(-2)	2.8
10	3.0(-2)	3.5
11	9.0(-3)	3.8
12	2.3(-3)	4.0
13	5.8(-4)	4.0

Table 6.18: Convergence study of MLAT code using a 2 grid system, $x_{\max} = 4$

Matching First Derivative at interface

l	$e\,(l)$	$Q\,(l, l+1)$
1	2.7(-2)	2.9
2	8.0(-3)	3.4
3	2.2(-3)	3.7
4	5.9(-4)	3.7
5	1.7(-4)	3.5
6	5.4(-5)	3.2
7	1.9(-5)	2.8
8	7.6(-6)	2.5

Table 6.19: Convergence study of MLAT code using a 2 grid system, $x_{\max} = 4$, with the addition of an extra grid point.

l	$e\,(l)$	$Q\,(l, l+1)$
1	2.3(-2)	2.1
2	7.4(-3)	3.4
3	1.9(-3)	3.8
4	4.8(-4)	4.0
5	1.1(-4)	4.3
6	2.3(-5)	4.6
7	4.0(-6)	5.9
8	1.1(-6)	3.8
x12	1.4(-2)	
x23	1.9(-3)	7.4
x34	1.6(-4)	11.9
x45	1.1(-5)	14.5
x56	6.8(-7)	16.1

Table 6.20: Convergence study of MLAT code using a 2 grid system, $x_{\max} = 4$

Smoothing τ_h^H at the interface

l	$e\,(l)$	$Q\,(l, l+1)$
1	1.6(-1)	2.1
2	7.8(-2)	2.9
3	2.7(-2)	3.5
4	7.7(-3)	3.9
5	2.0(-3)	4.0
6	5.0(-4)	3.9
7	1.3(-4)	4.0
8	3.2(-5)	4.0
x12	4.9(-2)	4.9
x23	1.0(-2)	8.0
x34	1.2(-3)	12.0
x56	7.2(-6)	15.7
x67	4.6(-7)	16.0
x78	2.9(-8)	16.0

Table 6.21: Convergence study of MLAT code using a 10 grid system, $x_{\max} = 512$

Smoothing τ_h^H at the interface

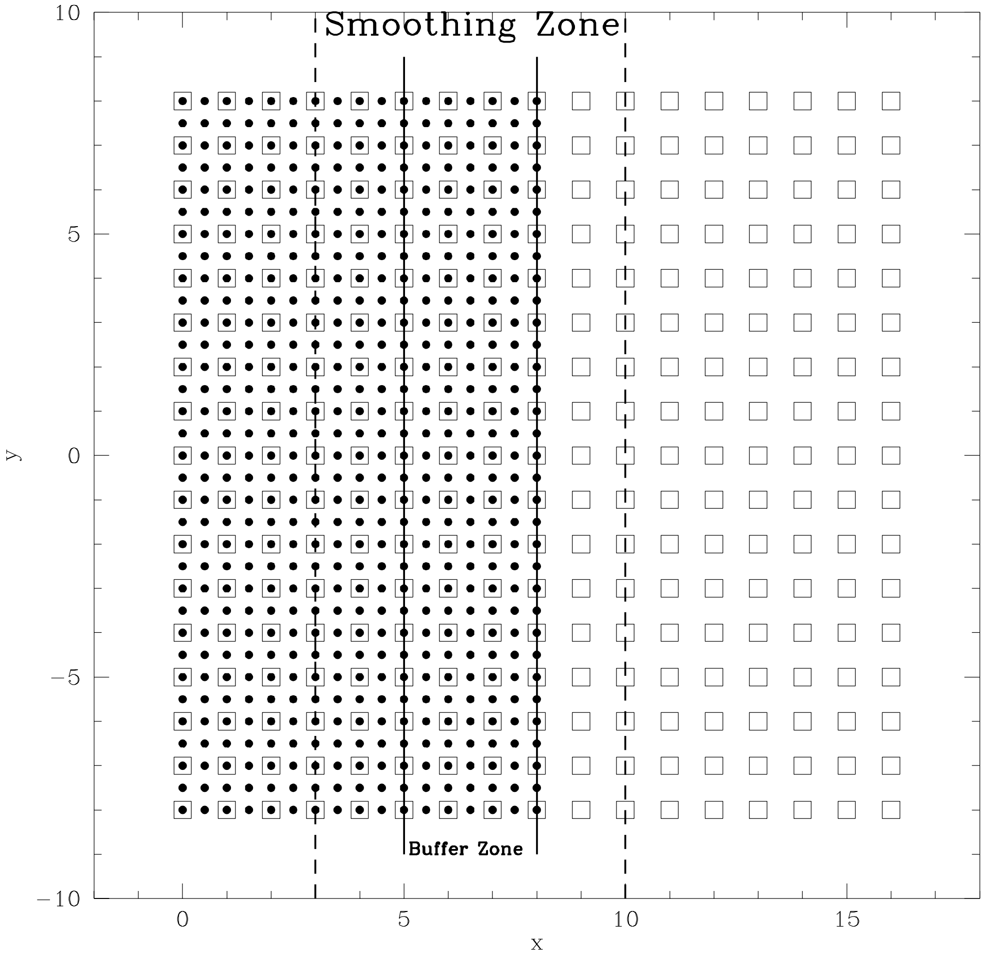
ν	$e(\nu)$	$Q(\nu, \nu+1)$
1	7.8(-2)	1.7
2	4.5(-2)	3.5
3	1.3(-2)	3.9
4	3.4(-3)	4.0
5	8.5(-4)	4.0
x12	3.6(-2)	17.1
x23	2.1(-3)	13.4
x34	1.7(-4)	15.5
x45	1.1(-5)	16.7

Table 6.22: Convergence study of 2D MLAT code using a 2 grid system, $x_{\max} = 512$

Smoothing and modifying τ_h^H at the interface

l	$e(l)$	$Q(l, l+1)$
2	1.3(-2)	2.5
3	5.2(-3)	9.6
4	5.4(-4)	2.8
5	1.9(-4)	3.7
6	5.2(-5)	4.0
7	1.3(-5)	4.0
8	3.3(-6)	4.0
x23	4.6(-3)	9.2
x34	5.0(-4)	6.0
x45	8.3(-5)	22.4
x56	3.7(-6)	16.0
x67	2.3(-7)	16.0
x78	1.4(-8)	16.0

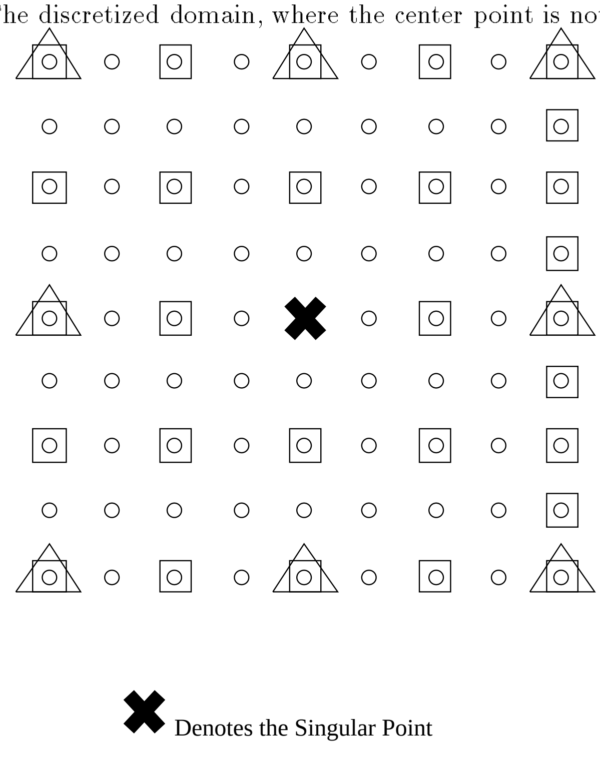
Figure 6.38: 2D model problem grid structure with 1 interface



6.5 Fast convergence for 1D problem with non-contained grids

In this section we will develop transfers designed to yield fast convergence when we use non-contained grids. Our adaptive structures typically look like the one shown in Figure 6.39, which is a 2d example. Now $r = 0$, (the center point of the figure), will generally not be part of the computational domain at any level of discretization. This leads to the property that points on fine grids in the vicinity of $x[j] = a$ will not be properly contained by a coarser grid. We refer to this property as non-containment or incomplete nesting. For a 1-dimensional problem a typical grid structure looks like that shown in Figure 4.3. There the X's are locations which lie outside of the computational domain and the filled points are points where a Robin boundary condition will be used.

Figure 6.39: The discretized domain, where the center point is not in the domain



Our model problem for this section is

$$\begin{aligned}
\frac{\partial^2 u(x)}{\partial x^2} &= f(x) \quad a < x < x_{\max} \\
\frac{\partial u}{\partial x} - \frac{u}{2a} &= g, \quad x = a \\
u(x_{\max}) &= u(x = x_{\max}) .
\end{aligned} \tag{6.45}$$

These equations are finite differenced to

$$\frac{u[j+1] - 2u[j] + u[j-1]}{h^2} = f[j] \quad j = j_{\min} \cdots n-1, \tag{6.46}$$

and

$$\frac{u[j+1]-u[j]}{h} = \frac{u[j]}{2a} + g \quad j = j_{\min} \tag{6.47}$$

where $j_{\min}$ is the point where the boundary condition will be imposed. We will use

$$\begin{aligned} u(x) &= 1 + \frac{a}{x} \\ f(x) &= \frac{2a}{x^3} \\ g &= -\frac{2}{a} \end{aligned} \tag{6.48}$$

We use fourth-order deferred correction at the boundary, where the first-order and fourth-order boundary operators are:

$$B_1^h u^h[j] = \frac{1}{h}\left(u[j+1] - u[j] - \frac{u[j]}{2a}\right) \quad j = j_{\min} \tag{6.49}$$

and

$$B_4^h u^h[j] = u_\chi - \frac{u_\sigma}{2a} \tag{6.50}$$

where

$$\begin{aligned} u_\sigma = \frac{1}{24}\Big(& - 32sh^3u[j_{\min}+3] + 4s^4u[j_{\min}+3] - 4s^4u[j_{\min}+1] \\ & + s^4u[j_{\min}] + 6s^4u[j_{\min}+2] + s^4u[j_{\min}+4] \\ & - 28s^3hu[j_{\min}+3] - 36s^3hu[j_{\min}+1] + 48s^3hu[j_{\min}+2] \\ & + 6s^3hu[j_{\min}+4] - 56s^2h^2u[j_{\min}+3] - 104s^2h^2u[j_{\min}+1] \\ & + 35s^2h^2u[j_{\min}] + 114s^2h^2u[j_{\min}+2] + 11s^2h^2u[j_{\min}+4] \\ & + 10s^3hu[j_{\min}] + 6sh^3u[j_{\min}+4] \\ & + 24h^4u[j_{\min}] + 72su[j_{\min}+2] \\ & - 96sh^3u[j_{\min}+1] + 50su[j_{\min}]h^3\Big)h^{-4} \end{aligned} \tag{6.51}$$

and

$$\begin{aligned} u_\chi = -\frac{1}{12}h^{-4}\Big(& - 42s^2hu[j_{\min}+3] - 54s^2hu[j_{\min}+1] + 72s^2hu[j_{\min}+2] \\ & + 9s^2hu[j_{\min}+4] - 56sh^2u[j_{\min}+3] - 104sh^2u[j_{\min}+1] \\ & + 35sh^2u[j_{\min}] + 114sh^2u[j_{\min}+2] + 11sh^2u[j_{\min}+4] \\ & + 36h^3u[j_{\min}+2] + 25h^3u[j_{\min}] - 8s^3u[j_{\min}+2] \\ & - 8s^3u[j_{\min}+1] + 2s^3u[j_{\min}] + 12s^3u[j_{\min}+2] \\ & + 2s^3u[j_{\min}+4] - 16h^3u[j_{\min}+3] - 48h^3u[j_{\min}+1] \\ & + 3h^3u[j_{\min}+4] + 15s^2hu[j_{\min}]\Big) \end{aligned} \tag{6.52}$$

with

$$s = x\,(j_{\min}) - a. \tag{6.53}$$

We use a technique similar to the one described by Figure 6.27 to apply deferred correction (using the above operators) at the boundary point $j = j_{\min}$. The complete algorithm is shown in Figure 6.40 and we note that we are *not* using a FMG scheme. We again relax the boundary as

$$u[j] := u[j] + \alpha\,(u^*[j] - u[j]) \tag{6.54}$$

but now we use

$$\begin{aligned} \alpha &= \alpha_1 && \text{for non contained grids} \\ &= \alpha_2 && \text{for contained grids.} \end{aligned}$$

Thus, distinct boundary relaxation factors, α_1 and α_2 are again used for cases where the boundary grid point is not contained or is contained, respectively, by the parent grid.

Figure 6.40: The V-cycle routine used with interfaces

```
PROCEDURE VCYCLE_FAS( ncycle, l_max, pre, pst , order)
   DOcycle= 1 ,ncycle
      DOl = l_max, l_coarse + 1
         FIND_INNER_BOUNDARY( j_min )
         IF(l ≠ l_max OR cycle= 1 THEN
            DOp = 1 , pre
               RELAX_INTERIOR, j = j_min + 1 ··· n − 1
               α = PICK_ALPHA( α_1, α_2 )
               RELAX_BOUNDARY, j = j_min
            END DO
         END IF
         IF ( l = l_max ) THEN
            Apply deferred correction
         END IF
         τ_h^H[J] = L^H I_h^H u^h[j] − I_h^H L^h u^h[j[,     j = j_min + 1 ··· n − 1
         τ_h^H[J_min] = B_1^H I_h^H u^h[j_min] − I_h^H B_1^h u^h[j_min]
         f^H[J] = I_h^H f^h[j] + τ_h^H[J]     Correct the rhs
         g^H[J_min] = I_h^H g^h[j_min] + τ_h^H[J_min]
      END DO
      SOLVE(L^H u^H = f^H, and B_1^H u^H = g^H)
      DOl = l_coarse, l_max − 1
         u^h[j] := u^h[j] + I_H^h (u^H[j] − I_h^H u^h[J]) ,     j = j_min + 1 ··· n − 1
         u^h[j_min] := u^h[J_min] + I_H^h (u^H[J_min] − I_h^H u^h[j_min]) ,     j = j_min
         DOq = 1 , pst
               RELAX_INTERIOR
               α = PICK_ALPHA( α_1,α_2 )
               RELAX_BOUNDARY, j = j_min
         END DO
      END DO
   END DO
END PROCEDURE
```

6.5.1 Test runs

In our test runs, we always use 2 pre-smoothing sweeps on the finest level and 1 pre- and 1 post-smoothing sweep in all other cases. We also fix $\alpha_1 = 1$ and $\alpha_2 = 0$ for most of the runs, and stop the V-cycles only when the ℓ_1 norm of the interior residuals has reached the level of machine precision.

Our chief concern is to ensure that we get extrapolatable results and we begin with a case where all the grids are properly contained (completely nested). We first compare the convergence behavior of deferred-corrected and non-corrected results. Tables (6.23), (6.24) clearly show that we need to use deferred correction in order to get extrapolatable results.

We now consider the case when none of the grids are properly contained. For this case we use $a = 0.12506103515625$ and a coarsest-level grid spacing $h_{\text{coarse}} = 0.25$. Table 6.25 shows that we again get extrapolatable results.

Since we can get extrapolatable results, we are now in a position to investigate the convergence rate of this algorithm. For these tests we change the value of a which has the effect of changing the number of properly-contained grids. We again use fourth-order deferred correction of the boundary condition and set $x_{\max} = 2$ and $h_{\text{coarse}} = 1/2$ for all tests. Our goal is to make sure that the convergence rate remains approximately constant no matter how many grids are not properly contained. Also, in these tests, we experiment with the restriction and prolongation operators for the function values in the vicinity of the boundary.

We begin by running a benchmark where all the grids are properly contained. Thus we set $a = h_{\text{coarse}}$ so that $x^h(j_{\min}) = x^{2h}(j_{\min}) = x^{4h}(j_{\min}) = \cdots$. Table 6.26 shows the convergence rate, ρ, for levels 2 through 8. We see that the convergence rate remains constant after about the fifth level, at which point the problem is being solved in about one cycle.

For the next four tests, we will use the following transfers,

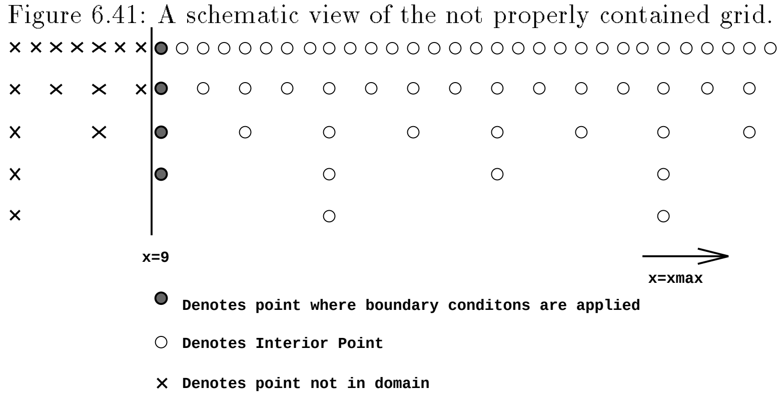
$$\begin{aligned} u^H[J_{\min}] &= I_h^H u^h[j_{\min}] = u^h[j_{\min}] \\ u^h[j_{\min}] &= I_H^h u^H[J_{\min}] = u^H[J_{\min}] \end{aligned} \tag{6.55}$$

where $j_{\min}$ and $J_{\min}$ are the grid locations where we apply the boundary conditions on the fine and coarse grids respectively. All the interior values on the grids are transferred with the usual restrictions and prolongations:

$$\begin{aligned} u^H[J] = I_h^H u^h[j] &= \frac{1}{2}u^h[j] + \frac{1}{4}\left(u^h[j-1] + u^h[j+1]\right) \\ u^h[j] = I_H^h u^H[J] &= u^H[J] \qquad 2J-1=j \\ &= \frac{1}{2}\left(u^H[J-1] + u^H[J+1]\right) \qquad 2J-1 \neq j \end{aligned} \tag{6.56}$$

For the first 3 of the 4 tests we will also use $\alpha_1 = 1.0$ and $\alpha_2 = 0.0$. In the first test only one grid is not contained (see Figure 6.41). The convergence test for these runs is shown

Figure 6.41: A schematic view of the not properly contained grid.



in Table 6.27, where $a = h_{\text{coarse}}/2$. We see that the convergence rate drops dramatically from the test case. We also see that the convergence rate gets better as l is increased since the number of grids that are properly contained increases. For the next test, we set $a = h_{\text{coarse}}/16$ which means that the four coarsest grids are not contained. The results of this run are shown in Table 6.28 where again we see another dramatic decrease in the convergence rate. Finally, we set $a = h_{\text{coarse}}/2048$ so that all grids are not-contained. As expected the results shown in Table 6.29 show another dramatic decrease in the convergence rate.

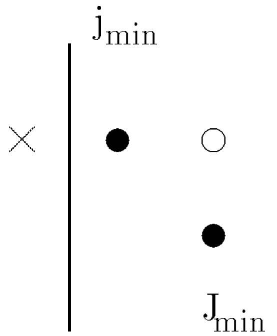
Our goal is to build a fast solver, which means that we would like the convergence rate, ρ, to be at least -0.1.[3] We must therefore use improved transfers around the boundaries. Through experimentation we have found that the point where transfers need to be modified is not at the boundary point but at the interior point immediately adjacent to the boundary point. Figure 6.42 shows a detailed view of the grid structure near the boundary. We originally used the following transfer for the values near the boundary:

$$\begin{aligned}
u^H[J_{\min}+1] &= I_h^H u^h[j_{\min}+3] = \frac{1}{2} u^h[j_{\min}+3] \\
&+ \frac{1}{4}\left(u^h[j_{\min}+2] + u^h[j_{\min}+4]\right) \\
u^h[j_{\min}+3] &= I_H^h u^H[J_{\min}+1] = u^H[J_{\min}+1] \\
u^h[j_{\min}+2] &= I_H^h u^H[J_{\min}+1] = \texttt{UNDEF} \\
u^h[j_{\min}+1] &= I_H^h u^H[J_{\min}+1] = \texttt{UNDEF}\ .
\end{aligned} \tag{6.57}$$

This left two function values undefined at the end of the transfer. We then tried to provide values at these points by means of linear interpolation of $u^h[j_{\min}]$ and $u^h[j_{\min}+3]$, but this procedure did not improve the convergence rate.

[3] We arbitrarily define this rate, which is much slower than all model multigrid algorithms.

Figure 6.42: A close-up view of the non contained grids



We started by experimenting with the transfers

$$u^H[J_{\min}+1] = \Xi[0]u^h[j_{\min}] + \sum_{i=1}^{10} \Xi[i]u^h[j_{\min}+i] \tag{6.58}$$

where, $0 \le \Xi[i] \le 1$, are the transfer coefficients. Through further experimentation we found the performance was best when we defined the value $u^H[J_{\min}+1]$ by:

$$\begin{aligned} u^H[J_{\min}+1] &= I_h^H u^h[j_{\min}+3] = 0.01u^h[j_{\min}+5] + 0.49u^h[j_{\min}+4] + \\ &+ 0.0u^h[j_{\min}+3] + 0.01u^h[j_{\min}+1] + 0.49u^h[j_{\min}+2]. \end{aligned} \tag{6.59}$$

These results suggest that residual at $J_{\min}+1$ is independent of the residual at $j_{\min}+3$ although $x[J_{\min}+1] = x[j_{\min}+3]$. The prolongation operators which we use are

$$\begin{aligned} u^h[j_{\min}+2] = I_H^h u^H[J_{\min}+1] &= -u^h[j_{\min}+4] + 2u^h[j_{\min}+3] \\ u^h[j_{\min}+1] = I_H^h u^H[J_{\min}+1] &= \frac{1}{2}\left(u^h[j_{\min}] + u^h[j_{\min}+2]\right). \end{aligned} \tag{6.60}$$

Note that the prolongation uses extrapolation to determine the value at $j_{\min}+2$. These operators are used for all functions which are transferred from one discretization level to another.

The convergence rate obtained when we use these operators is shown in Table 6.30 and we see that it is much faster than the original algorithm for the case when all grids are not properly contained. However, the improved convergence rate for the non-contained case is still much slower than that for the case when all grids are properly nested. We have, however, been able to reduce the residuals by one order of magnitude for every four work units, which we believe is an excellent convergence rate in comparison with the other traditional iterative solvers.

Table 6.23: 1D FMG FAS algorithm with inner boundary condition imposed.

Deferred correction not used

l	$e\,(l)$	$Q\,(l,l+1)$
2	4.7(-2)	2.0
3	2.4(-2)	2.0
4	1.2(-2)	2.0
5	6.1(-3)	2.0
6	3.0(-3)	2.0
7	1.5(-3)	2.0
8	7.6(-4)	2.0

Table 6.24: 1D FMG FAS algorithm with inner boundary condition imposed, with deferred correction to fourth order

l	$e\,(l)$	$Q\,(l,l+1)$
2	4.1(-3)	4.5
3	9.2(-4)	4.2
4	2.2(-4)	4.1
5	5.3(-5)	4.0
6	1.3(-5)	4.0
7	3.3(-6)	4.0
8	8.1(-7)	4.0
x23	2.1(-4)	8.1
x34	2.6(-5)	11.3
x45	2.3(-6)	13.5
x56	1.7(-7)	14.6
x67	1.2(-8)	15.4
x78	7.8(-10)	16.0

The last test we try varies α_1 to see if we can improve the convergence rate. (Recall that α_1 is the boundary smoothing factor for a grid which is not properly contained by its parent). Table 6.31 shows the convergence rate as a function of α_1 for the case $l = 8$. Clearly, the convergence rate is quite insensitive to the value of α_1 as long as $\alpha_1 < 2$.

Table 6.25: 1D FMG FAS algorithm with inner boundary condition, deferred correction to fourth order, and non-contained grids.

l	$e\,(l)$	$Q\,(l, l+1)$
2	6.4(-2)	4.6
3	1.4(-2)	6.1
4	2.3(-3)	5.2
5	4.1(-4)	4.7
6	8.7(-5)	4.4
7	2.0(-5)	4.0
8	5.0(-6)	4.0
9	1.2(-6)	4.0
x23	4.0(-3)	2.2
x34	1.8(-3)	6.2
x45	2.9(-4)	10.0
x56	2.9(-5)	12.6
x67	2.3(-6)	14.7
x78	1.6(-7)	16.0
x89	9.7(-9)	16.0

Table 6.26: The convergence rate of the 1D FMG FAS algorithm with inner boundary condition, deferred correction to fourth order, and no non-contained grids.

l	$\rho\,(l)$
2	-1.4(+0)
3	-1.4(+0)
4	-1.7(+0)
5	-1.8(+0)
6	-1.9(+0)
7	-1.9(+0)
8	-2.0(+0)

Table 6.27: The convergence rate of the 1D FMG FAS algorithm with inner boundary condition, deferred correction to fourth order, and one non-contained grid.

l	$\rho\,(l)$
2	-7.7(-2)
3	-1.2(-1)
4	-1.5(-1)
5	-1.7(-1)
6	-1.8(-1)
7	-1.9(-1)
8	-1.9(-1)

Table 6.28: The convergence rate of the 1D FMG FAS algorithm with inner boundary condition, deferred correction to fourth order, and four non-contained grids.

l	$\rho(l)$
2	-2.7(-2)
3	-5.0(-2)
4	-6.4(-2)
5	-6.0(-2)
6	-6.0(-2)
7	-6.6(-2)
8	-7.0(-2)

Table 6.29: The convergence rate of the 1D FMG FAS algorithm with inner boundary condition, deferred correction to fourth order, and all grids not-properly contained.

l	$\rho(l)$
2	-3.3(-2)
3	-4.8(-2)
4	-5.3(-2)
5	-4.8(-2)
6	-4.8(-2)
7	-4.2(-2)
8	-4.0(-2)
9	-3.8(-2)

Table 6.30: 1D FMG FAS algorithm with experimental transfers, using deferred correction to fourth order and all grids not-properly contained.

l	$\rho(l)$
2	-7.7(-2)
3	-1.2(-1)
4	-1.7(-1)
5	-2.1(-1)
6	-2.3(-1)
7	-2.4(-1)
8	-2.5(-1)
9	-2.5(-1)
10	-2.5(-1)

Table 6.31: Survey of the effect of α_1 on the convergence rate for non-contained grids

Deferred correction used to fourth order, $l = 8$, all grids unaligned

$\alpha_1\,(l)$	$\rho\,(l)$
0.0(-0)	-2.38(-1)
1.0(-3)	-2.38(-1)
1.0(-2)	-2.39(-1)
5.0(-2)	-2.44(-1)
1.0(-1)	-2.53(-1)
2.0(-1)	-2.46(-1)
1.0(+0)	-2.47(-1)
2.0(+0)	-2.49(-1)
3.0(+0)	-7.43(-2)

6.6 1D extrapolatable adaptive MLAT with blended boundary condition and outer boundary condition

Before we combine all of the elements of the last three sections to form a single algorithm, we wish to explore one more aspect of equation (6.47). Since this equation models the boundary condition we use at the interior (hole) boundaries in 2 and 3 dimensions, we would like to understand how to effectively treat it in the contexts of mesh refinement and multigrid.

Choptuik[14] pointed out that since this equation will be used on a extremely large range of refinement levels, the performance of the solver could suffer since the very coarse grids will not provide reasonable approximations of the function derivative near the boundary. Our solution to this problem is to use a "blended" boundary condition. Recall that the condition we pose at the boundary is:

$$\frac{\partial u}{\partial x} - \frac{u}{2a} = g, \qquad x = a, \tag{6.61}$$

which we now represent as

$$B_\beta u\left(x\right) = g_1 \qquad x = a \tag{6.62}$$

where

$$g_1 = -\frac{2}{a}. \tag{6.63}$$

We only want to apply this Robin boundary condition on the finer levels. On the coarser levels, where we will not be able to get a good approximation for the derivative, we will apply a Dirichlet boundary condition

$$B_\gamma u\left(x\right) = g_2 \quad x = a, \tag{6.64}$$

where

$$\begin{aligned} B_\gamma u\left(x\right) &= u\left(x\right) \quad x = a \\ g_2 &= I_h^H u^h\left(x\right). \end{aligned} \tag{6.65}$$

This last condition just sets the coarse grid value, $u^H\left(a\right)$ to the corresponding fine grid value, $u^h\left(a\right)$. The blended boundary condition is then given by:

$$\left(\beta B_\beta + \left(1-\beta\right) B_\gamma\right) u = \beta g_1 + \left(1-\beta\right) g_2 \tag{6.66}$$

where β is a function of the discretization level l such that $\beta\left(l_{\max}\right) = 1$ (so that we recover the true Robin condition on the finest level) and $\beta\left(0\right) = 0$ (so that the coarsest problem has Dirichlet conditions). For brevity we can then rewrite equation (6.66), as

$$Bu = g \tag{6.67}$$

where

$$B = \beta B_{\beta} + (1-\beta)\, B_{1-\beta} \tag{6.68}$$

and

$$g = \beta g_1 + (1-\beta)\, g_2. \tag{6.69}$$

The smoothing operator on the boundary is now defined as

$$u[j_{\min}] = \frac{-2a\beta u[j_{\min}+1] + 2ah\beta g_1 + 2ah\,(1-\beta)\, g_2}{2ah\,(1-\beta) - 2a\beta - h\beta}. \tag{6.70}$$

We use injections for our boundary transfers but the functions g_1 and g_2 are treated separately in these transfers. Specifically, we use:

$$\begin{aligned} g_1^H &= g_1^h + \tau_h^H{}_{\beta} \\ g_2^H &= g_2^h + \tau_h^H{}_{(1-\beta)} \end{aligned} \tag{6.71}$$

where $\tau_h^H{}_{\beta}$ and $\tau_h^H{}_{(1-\beta)}$ are given by

$$\begin{aligned} \tau_h^H{}_{(1-\beta)} &= 0 \\ \tau_h^H{}_{\beta} &= B_{\beta}^H I_h^H u^h - I_h^H B_{\beta}^h u^h. \end{aligned} \tag{6.72}$$

We will now study the effects of varying $\beta\,(l)$. For these experiments we fix $a = h_{\text{coarse}}/16$, $h_{\text{coarse}} = 1/2$ and $x_{\max} = 2$. Table 6.32 shows the effect of varying $\beta\,(l)$ on a 10 level system. To determine the best β's we ran a binary search through the values of $\beta\,(l)$ which ranged from zero to one in 0.1 increments. The bold face numbers were the best parameters that we were able to find.

We can not claim that our best $\beta\,(l)$ yields the best possible convergence using the blended approach since, obviously, we have not tried every conceivable combination. What we do see in this table, though, is that the convergence rate can be improved from -0.29 to -0.36 which means that we can improve the convergence rate by approximately one sweep per order of magnitude of residual reduction. This amounts to about a 25% reduction in computational cost.

When we run tests with $x_{\max} \gg h_{\text{coarse}}$, we do not seem to be able to get results which are much better than those obtained without the use of a blended boundary condition. However, when we apply this technique to our 2- and 3-dimensional problems, we hope that the blended boundary technique will allow us to use non-diagonally dominant stencils on the finest grids, while solving coarse-grid systems which have all-Dirichlet conditions.

We have now introduced all the techniques we need to get a fast, extrapolatable solver using the Robin conditions on all boundaries. A 1-D algorithm which incorporates these features is shown in Figure 6.43. This algorithm works when only one grid is defined per level.

Again, our model problem is:

$$\begin{aligned}
\frac{\partial^2 u(x)}{\partial x^2} &= f(x) \quad a < x < x_{\max} \\
\frac{\partial u}{\partial x} - \frac{u}{2a} &= g, \quad x = a \\
\frac{\partial u}{\partial x} &= \frac{1-u}{x} \quad x = x_{\max}
\end{aligned} \tag{6.73}$$

$$\tag{6.74}$$

our finite difference approximation is

$$\begin{aligned}
&h^{-2}(u[j+1] - 2u[j] + u[j-1]) = f[j] \quad j = j_{\min} + 1 \cdots n - 1 \\
&u^{**}[j_{\min}] = \frac{-2a\beta u(j_{\min}+1) + 2ah\beta g_1 + 2ah(1-\beta) g_2}{2ah(1-\beta) - 2a\beta - h\beta} \\
&u[j] := u[j] + \alpha(u[j]^{**} - u[j]) \quad j = j_{\min} \\
&u[j]^* = \frac{1}{1+\frac{h}{x_{\max}}}\left(\frac{h}{x_{\max} + u[j-1] + hg}\right) \quad j = n \\
&u[j] := u[j] + \alpha(u[j]^* - u[j]) \quad j = n,
\end{aligned} \tag{6.75}$$

and we use the model functions

$$\begin{aligned}
u(x) &= 1 + \frac{a}{x} \\
f(x) &= \frac{2a}{x^3} \\
g(x=a) &= -\frac{2}{a}
\end{aligned} \tag{6.76}$$

to examine the convergence behavior of our algorithm

Figure 6.44 shows a graph of the log of the absolute errors, $|e|$, for the raw solution and the extrapolated solution. (Again, the lines here are separated by a spacing of $\log_{10}(2)$). We clearly see in this graph that the open symbols are displaced by 2 lines, while the filled symbols are displaced by 4 lines. This demonstrates that the basic solution is second order, while the extrapolated solution is fourth order. For this run we only used one refinement, and placed the inner and outer boundaries at $a \approx \frac{1}{4}$ and $x_{\max} = 2.0$, respectively. The placement of the inner boundary is such that no grid point at any level will ever be coincident with it. To make the graph more readable, we plot every thirty-second point at each level.

As previously claimed, our algorithm works with one refinement, it should work with more. Figure 6.45 shows the results from a run where $a \approx \frac{1}{4}$ and $x_{\max} = 128.0$. Here we used a 5-point coarsest grid. Since we use so many interfaces, it is better to show the results on a log-log scale.

In summary, our 1-d algorithm produces extrapolatable solutions of 1-d analogues of the equations that we will solve for the initial value problem for two black holes. We did not study convergence rates in this section since we still do not fully understand the stopping criteria when we smooth the relative truncation error estimate around the interfaces. We have clearly presented a path by which we can build multi-dimensional versions of the one dimensional algorithm. However, some additional work will be needed to determine appropriate multi-dimensional transfers for our irregular interior boundaries.

Figure 6.43: The 1D FAS V-cycle routine which contains the inner and outer boundary conditions

PROCEDURE VCYCLE_FAS_1D_EXTRAP(ncyc, $l_{\max}$, pre, pst, order)
 DOcycle= 1 ,ncycle
 Cycle up to the coarsest level
 DOl $= l_{\max}, l_{\text{coarse}} + 1$
 FIND_INNER_BOUNDARY($j_{\min}$)
 IF(l $\neq l_{\max}$ ORcycle= 1) THEN
 FIND_INTERFACE(j_max)
 j_max = FIND_INTERFACE *will find the interface point*
 DOp = 1 , pre
 RELAX_INTERIOR($u^h[j] = f^h[j]$) $\quad j = j_{\min} \cdots n-1$
 IF(gmax(lhead(l))=$x_{\max}$) THEN
 RELAX_OUTER_BOUNDARY($u^h[n] = g^h[n], \alpha_0$) j=n
 END IF
 $\alpha =$ PICK_ALPHA($alpha_1$, $alpha_2$)
 RELAX_INNER_BOUNDARY($u^h[j_{\min}] = g^h[j_{\min}], \alpha,$
 β(lhead(l))), $\quad j = j_{\min}$
 END DO
 END IF
 IF (GMAX(lhead(l))= $x_{\max}$ AND
 gmax(gpch(lhead(l))) $\neq x_{\max}$) THEN
 Now apply deferred correction
 END IF
 IF (l $= l_{\max}$) THEN
 Now apply deferred correction
 $g^h[j_{\min}] := g + B_4^h(u[j_{\min}]) - B_1^h(u[j_{\min}])$
 END IF

Figure 6.43 continued

```
    τ_h^H[J] = L^H I_h^H u^h[j] − I_h^H L^h u^h[j],     J = J_min ··· J_max-1
    τ_h^H(j) = 0  ,   j =j_max··· n
    set τ_h^H[j] to zero where it is not defined
    IF( gmax(lhead(l))=x_max) THEN
       τ_h^H[n] = Z_1^H I_h^H u^h[n] − I_h^H B_1^h u^h[n]
       g^H[N] = I_h^H g^h(n) + τ_h^H(N)
    END IF
    τ_h^H[J_min] = B_1^H I_h^H u^h[j_min] − I_h^H B_1^h u^h[j_min]
    g^H[J_min] = I_h^H g^h[j_min] + τ_h^H[j_min]
    f^H[J] = I_h^H f^h[j] + τ_h^H[j]
    When j ≥ j_max then determine f^H[J] from the analytical rhs
  END DO
  SOLVE(L^H u^H = f^H, and Z_1^H u^H = g^H), and B_1^H u^H = g^H
 DOl = l_coarse, l_max − 1
    u^h[j] := u^h[j] + I_H^h (u^H[j] − I_h^H u^h[j])  ,    j = j_min ··· j_max-1
    IF( GMAX(lhead(l))=x_max) THEN
       u^h[n] := u^h[n] + I_H^h (u^H[n] − I_h^H u^h[n])  ,    j = n
    END IF
    u^h[j_min] := u^h[J_min] + I_H^h (u^H[J_min] − I_h^H u^h[j_min]),    j = j_min
    DOq = 1 , pst
          RELAX_INTERIOR(u^h[j], f^h[j])    j = 2 ··· n − 1
          IF( gmax(lhead(l))=x_max) THEN
             RELAX_OUTER_BOUNDARY(u^h[n], g^h[n], α_0)    j=n
          END IF
          α = PICK_ALPHA( α_1, α_2 )
          RELAX_INNER_BOUNDARY(u^h[j_min] = g^h[j_min], α,
                                       β(lhead(l))),    j = j_min
    END DO
  END DO
 END DO
END PROCEDURE
```

Table 6.32: 1D FMG FAS algorithm with blended boundary condition, and experimental transfers

Deferred correction used to fourth order, all grids unaligned, $l = 10$, $x_{\max} = 2$

$\beta\,(0)$	$beta\,(1)$	$\beta\,(2)$	$\beta\,(3)$	$\beta\,(4)$	$\beta\,(5)$	$\beta\,(6)$	$\beta\,(7)$	$\beta\,(8)$	$\beta\,(9)$	ρ
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-2.88(-1)
1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.0	-3.43(-1)
1.0	0.9	0.8	0.7	0.6	0.0	0.4	0.3	0.3	0.0	-3.10(-3)
1.0	0.9	0.8	0.7	0.6	1.0	0.4	0.3	0.3	0.0	-3.19(-1)
1.0	0.9	0.8	0.7	0.6	0.4	0.4	0.3	0.3	0.0	-3.46(-1)
1.0	0.9	0.8	0.7	0.6	0.3	0.4	0.3	0.3	0.0	-3.57(-1)
1.0	0.9	0.8	0.7	0.6	0.3	0.0	0.3	0.3	0.0	-1.56(-1)
1.0	0.9	0.8	0.7	0.6	0.3	1.0	0.3	0.3	0.0	-3.41(-1)
1.0	0.9	0.8	0.7	0.6	0.3	0.4	0.0	0.3	0.0	-2.11(-1)
1.0	0.9	0.8	0.7	0.6	0.3	0.4	1.0	0.3	0.0	-3.58(-1)
1.0	0.9	0.8	0.7	0.6	0.3	0.4	1.0	0.3	0.0	-3.58(-1)
1.0	0.9	0.8	0.7	0.6	0.3	0.4	1.0	0.0	0.0	-2.65(-1)
1.0	0.9	0.8	0.7	0.6	0.3	0.4	1.0	1.0	0.0	-3.54(-1)
1.0	**0.9**	**0.8**	**0.7**	**0.6**	**0.3**	**0.4**	**1.0**	**0.6**	**0.0**	**-3.62(-1)**
1.0	0.9	0.8	0.7	0.6	0.3	0.4	1.0	0.6	0.5	-2.78(-1)
1.0	0.9	0.8	0.7	0.6	0.3	0.4	1.0	0.6	1.0	-2.78(-1)

Table 6.33: 1D FMG FAS algorithm with inner and outer boundary condition and blended boundary

$a = \frac{1}{e}$, $x_{\max} = 2.0$

l	$E_1\,(l)$	$Q\,(l, l+1)$
2	1.3(-2)	2.5
3	5.2(-3)	9.6
4	5.4(-4)	2.8
5	1.9(-4)	3.7
6	5.2(-5)	4.0
7	1.3(-5)	4.0
8	3.3(-6)	4.0
x34	8.8(-4)	9.2
x45	2.0(-4)	22.4
x56	2.4(-5)	16.0
x67	1.5(-6)	16.0
x78	9.4(-8)	16.0
x89	5.9(-9)	16.0
x910	3.7(-10)	16.0
x1011	2.3(-11)	16.0
x1112	1.4(-12)	16.0

Figure 6.44: The errors for extrapolated and non extrapolated results using 2 adaptive refinement levels.

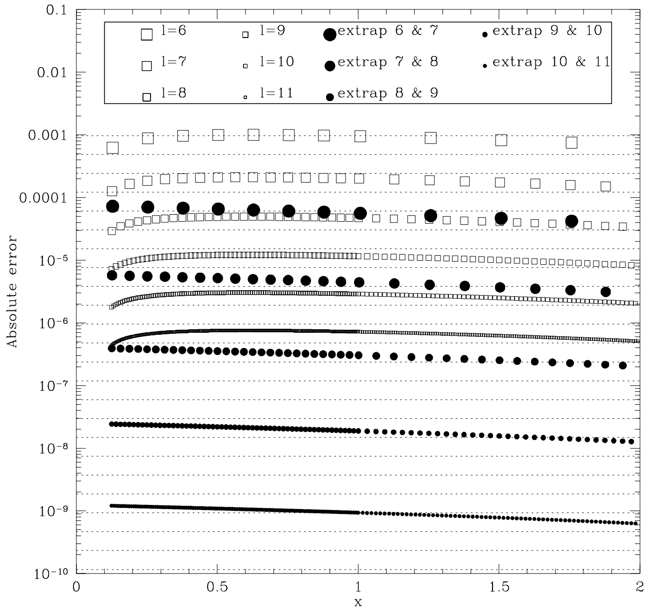
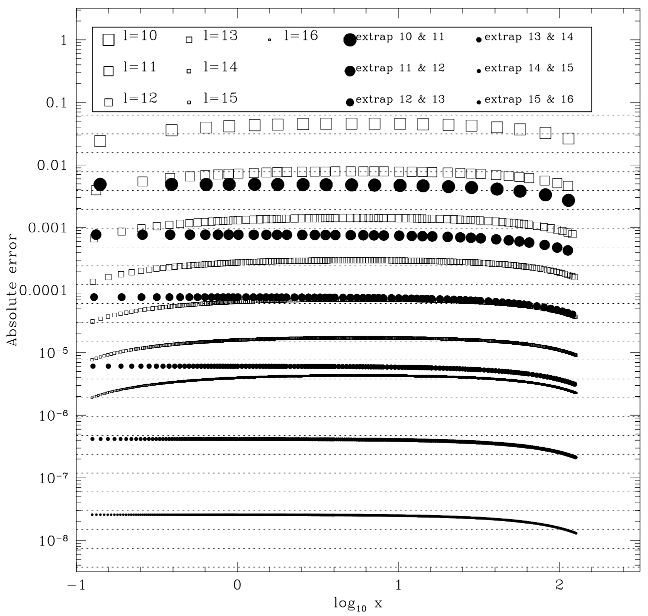


Figure 6.45: The errors for extrapolated and non extrapolated results using 8 adaptive refinement levels.



6.7 Two dimensional model problem with non-aligned grids

In this section we design a multigrid code which solves the equations

$$
\begin{aligned}
\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} &= f(x,y) \qquad x_{\min} \le x \le x_{\max},\, y_{\min} \le y \le y_{\max} \\
\frac{\partial u(x,y)}{\partial r} &= -\frac{u(x,y)}{2\,a} \quad r = a \\
u(x = x_{\max}, y) &= u(x_{\max}, y) \\
u(x, y = y_{\max}) &= u(x, y = y_{\max})
\end{aligned}
\tag{6.77}
$$

where $r = \sqrt{(x - c_x)^2 + (y - c_y)^2}$, and the center of the hole is at (c_x, c_y). We use a uniform Cartesian grid, and then exclude a roughly circular region from the computational domain as shown in Figure 6.46.

In order to fully describe our algorithm we first describe each major component separately. After we have presented all of the major routines we will then display the entire algorithm in figures (6.52) and (6.53).

6.7.1 Generation of the characteristic function

We define a characteristic function, $\texttt{char}(j,k)$, where

$$
\begin{array}{ll}
\texttt{char}^*(j,k) = 1 & r \ge a \\
\texttt{char}^*(j,k) = 0 & r < a \\
\texttt{char}(j,k) = 2 & \text{if } \texttt{char}^*(j,k) = 1 \text{ and} \\
& \Big(\texttt{char}^*(j+1,k) = 0 \text{ or } \texttt{char}^*(j-1,k) = 0 \text{ or} \\
& \texttt{char}^*(j,k+1) = 0 \text{ or } \texttt{char}^*(j,k-1) = 0\Big) \\
& \text{else } \texttt{char} = \texttt{char}^*
\end{array}
$$

Thus, in Figure 6.46, the $\cdot$, $\bullet$, and $\circ$ symbols label points where $\texttt{char}(j,k)$ is 0, 1, or 2 respectively. We use this characteristic function to encode which equations, if any, will be applied at the grid point with indices (j,k).

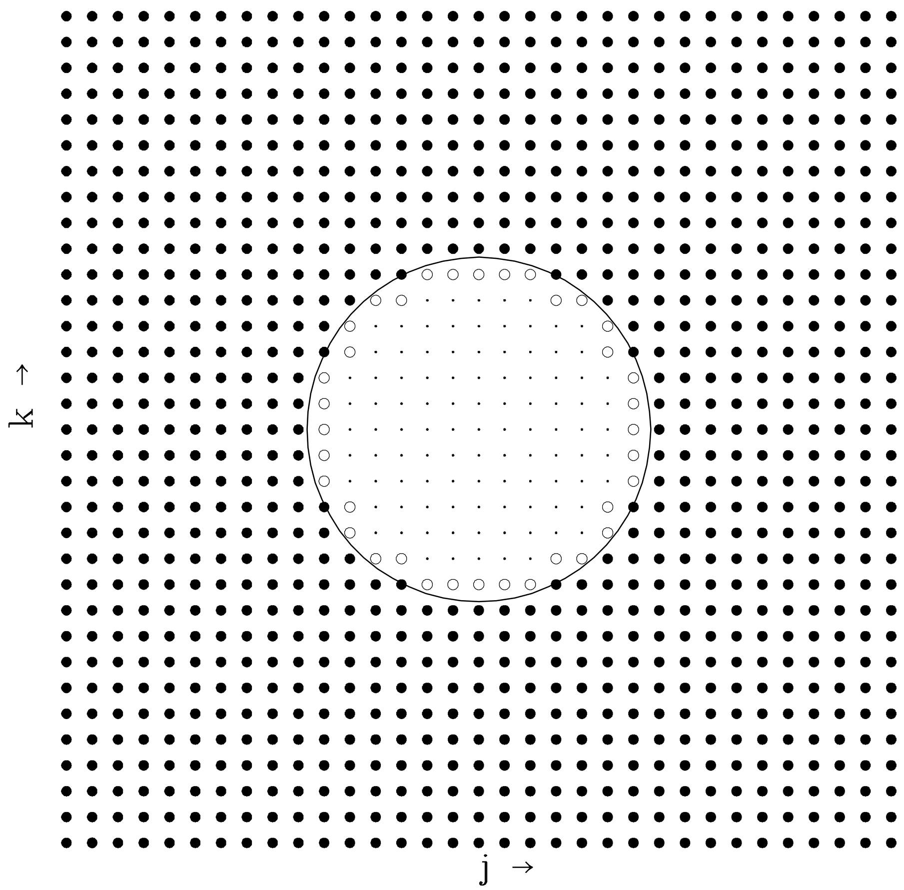
6.7.2 Smoothing

We use red-black Gauss-Seidel relaxation to smooth the residuals of the interior equations shown in the algorithm given in Figure 6.48.

The boundary smoothing is much more complicated than the interior smoothing. We only relax the boundary to first order, but care must be taken to determine which stencils are used on the boundaries. We pre-compute the stencils and their coefficients to ensure

Figure 6.46: The 2D computational domain.

In this figure, interior points are labeled with the • symbol, points where the Robin boundary condition will be applied are indicated by the ○ symbol, and points not in the computational domain are referenced by the · symbol



boundary smoothing remains inexpensive compared to interior smoothing. We now describe the boundary smoothing in several stages. [4]

Once we have determined `char` (j,k) we can precompute the coefficients for the various boundary stencils. We finite-difference the boundary equation to first order using

$$\frac{n_x\ \varrho}{h}\left(u[j+\varrho,k]-u[j,k]\right)+\frac{n_y\ \vartheta}{h}\left(u[j,k+\vartheta]-u[j,k]\right) \tag{6.78}$$

where ϱ and ϑ are either 1 or -1 depending upon if forward or backward differencing was used and

$$\begin{aligned} n_x &\equiv \frac{x-c_x}{r} \\ n_y &\equiv \frac{y-c_y}{r}. \end{aligned} \tag{6.79}$$

We then store the coefficients of the stencil in arrays κ and λ, defined by

$$\begin{aligned} \kappa &= \frac{n_x\ \varrho}{h} \\ \lambda &= \frac{n_y\ \vartheta}{h}. \end{aligned} \tag{6.80}$$

The other variables which we will store are: n_{P}, the number of points on the boundary, $\imath\left(n_{\mathrm{P}}\right)$, the indices j where `char` $(j,k)=2$, and $\jmath\left(n_{\mathrm{P}}\right)$, the indices k where `char` $(j,k)=2$.

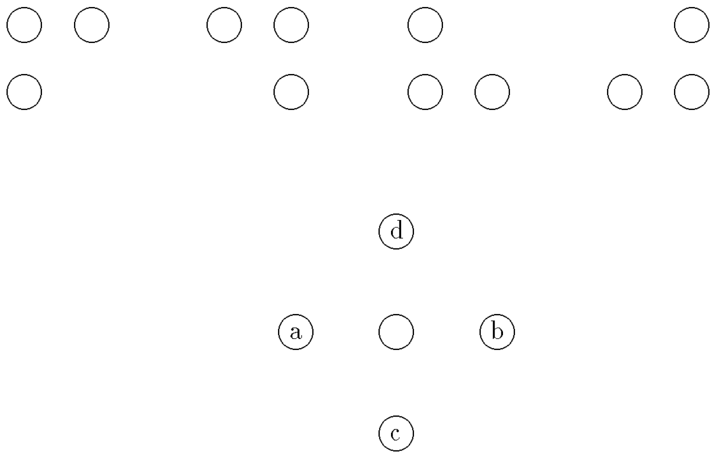
There are 4 first order stencils, shown in Figure 6.47, along with a five point stencil with the points being labeled as a,b,c and d. Since there is at least one point in the five point stencil not in the computational domain for the points surrounding the boundary, we will use the algorithm shown in Figure 6.49 to determine which points will be used for the first order stencil.

After we determine the lists containing the x and y indices of the boundary points ($\imath$ and $\jmath$), we define three additional arrays, $\mathrm{X}(n_{\mathrm{P}})$, $\mathrm{Y}(n_{\mathrm{P}})$ and $\theta\left(n_{\mathrm{P}}\right)$, which define the location of the physical boundary which is nearest to the various boundary grid points. Specifically:

$$\begin{aligned} \theta\left(n_{\mathrm{P}}\right) &= \arctan\left(\frac{y\left(\jmath\left(n_{\mathrm{P}}\right)-c_y\right)}{x\left(\imath\left(n_{\mathrm{P}}\right)-c_x\right)}\right) \\ \mathrm{X}\left(n_{\mathrm{P}}\right) &= c_x+a\ \cos\left(\theta\left(n_{\mathrm{P}}\right)\right) \\ \mathrm{Y}\left(n_{\mathrm{P}}\right) &= c_y+a\ \sin\left(\theta\left(n_{\mathrm{P}}\right)\right). \end{aligned} \tag{6.81}$$

[4] At this point, we must define what we mean by the term "boundary point". We define a boundary point as a point on the lattice in which we will impose the boundary condition. This point is in general *not* on the physical boundary, but rather $O\left(h\right)$ away from the boundary.

Figure 6.47: A view of the four first order stencils and a five point stencil.



After these arrays are defined, they are then sorted in order of ascending $\theta\,(n_{\mathrm{P}})$ and then we compute x and y components of the normals to the boundary using:

$$\begin{aligned} n_x\,(n_{\mathrm{P}}) &= \frac{\mathrm{X}\,(n_{\mathrm{P}}) - c_x}{a} \\ n_y\,(n_{\mathrm{P}}) &= \frac{\mathrm{Y}\,(n_{\mathrm{P}}) - c_y}{a}. \end{aligned}$$

At this point, κ and λ, the coefficients of the stencils where the boundary equation will be applied, can be determined.

The smoothing is now simply defined by the algorithm in Figure 6.50. We see here that we also store an array for the values of u along the boundary. This is done purely for ease of programming.

Figure 6.48: The interior smoothing algorithm used in 2D

PROCEDURE RELAX_INTERIOR(u , f , h , char)

This routine will smooth the interior points via Red-Black Gauss-Seidel

points = 0 *Keep track of the number of points smoothed on this grid*

$jacel = -4h^{-2}$ *Pre-compute the Jacobian of the system*

koff = 1 *Variable used for Red-Black Smoothing*

DO checker_board = 1 , 2 *Loop over the Red and Black points*

joff = koff *Variable used for Red-Black Smoothing*

DO k = 2,ny - 1 *Loop over all the ponts on the k axis*

DO j = ioff+1 , nx -1 , 2

$$rresl = h^{-2}\Big(u[j+1,k] + u[j-1,k] + u[j,k+1] + u[j,k-1] - 4\ u[j,k] - f[j,k]\Big)$$

$u[j,k] := u[j,k] - \mathtt{char}(j,k)\left(rresljacel^{-1}\right)$

Only change the points where $\mathtt{char}(j,k) = 1$

$r[j,k] = \mathtt{char}(j,k)\,(rresl)$

Define the running residual at this point

END DO

joff = 3 - joff

END DO

koff = 3 - koff

END DO

END PROCEDURE

Figure 6.49: The algorithm which determines the stencil to use.

```
PROCEDURE DETERMINE_STENCIL( n_P , ϱ, ϑ , j, k, char,
                                                    x, y, c_x, c_y )
   a = char(j − 1, k) , b = char(j + 1, k)
   c = char(j, k − 1) , d = char(j − 1, k + 1)
   IF ((a = 0) AND (c = 0)) THEN
      ϱ(n_P) = 1 , ϑ(n_P) = 1
   ELSE IF ((a = 0) AND (d = 0)) THEN
      ϱ(n_P) = 1 , ϑ(n_P) = −1
   ELSE IF ((b = 0) AND (c = 0)) THEN
      ϱ(n_P) = −1, ϑ(n_P) = 1
   ELSE IF ((b = 0) AND (d = 0)) THEN
      ϱ(n_P) = −1, ϑ(n_P) = −1
   ELSE IF (a = 0)THEN
      r1 = (x(j) − c_x)^2 + (y(k + 1) − c_y)^2 , r2 = (x(j) − c_x)^2 + (y(k − 1) − c_y)^2
      IF (r1 ≥ r2) THEN
         ϱ(n_P) = 1 , ϑ(n_P) = 1
      ELSE
         ϱ(n_P) = 1 , ϑ(n_P) = −1
      END IF
   ELSE IF (b = 0) THEN
      r1 = (x(j) − c_x)^2 + (y(k + 1) − c_y)^2 , r2 = (x(j) − c_x)^2 + (y(k − 1) − c_y)^2
      IF (r1 ≥ r2) THEN
         ϱ(n_P) = −1 , ϑ(n_P) = 1
      ELSE
         ϱ(n_P) = −1 , ϑ(n_P) = −1
      END IF
   ELSE IF (c = 0) THEN
      r1 = (x(j + 1) − c_x)^2 + (y(k) − c_y)^2 , r2 = (x(j − 1) − c_x)^2 + (y(k) − c_y)^2
      IF (r1 ≥ r2) THEN
         ϱ(n_P) = 1 , ϑ(n_P) = 1
      ELSE
         ϱ(n_P) = −1 , ϑ(n_P) = 1
      END IF
   ELSE
      r1 = (x(j + 1) − c_x)^2 + (y(k) − c_y)^2 , r2 = (x(j − 1) − c_x)^2 + (y(k) − c_y)^2
      IF (r1 ≥ r2) THEN
         ϱ(n_P) = 1 , ϑ(n_P) = −1
      ELSE
         ϱ(n_P) = −1 , ϑ(n_P) = −1
      END IF
   END IF
END PROCEDURE
```

Figure 6.50: The algorithm to smooth the boundary equations

```
PROCEDURE RELAX_BOUNDARY( u ,g, n_P , ı , ȷ , κ, λ ,
                                    ϱ , SFK )
   DO i = 1, n_P
      j = ı(i)
      k = ȷ(i)
      ub[i] = g[j, k] − κ  u[j + ϱ(i), k] + λ  u[j, k + ϑ(i)]
      u[j, k] = ub[i]
   END DO
 END PROCEDURE
```

6.7.3 Transfer operators

We use the following interior restriction and prolongation operators:

$$
\begin{aligned}
u^H[J,K] &= I_h^H u^h[j,k] \\
&= \frac{1}{8}\left(u^h[j-1,k]+u^h[j+1,k]+u^h[j,k-1]+u^h[j,k+1]\right) \\
&+ \frac{1}{2}u^h[j,k]
\end{aligned}
$$

and

$$
\begin{aligned}
u^h[j,k] &= u^H_{J,K} \\
u^h[j+1,k] &= \frac{1}{2}\left(u^H[J,K) + u^H[J+1,K]\right) \\
u^h[j,k+1] &= \frac{1}{2}\left(u^H[J,K) + u^H[J,K+1]\right) \\
u^h[j+1,k+1] &= \frac{1}{4}\Big(u^H[J,K] + u^H[J+1,K] \\
&+ u^H[J,K+1] + u^H[J+1,K+1]\Big).
\end{aligned}
$$

Conceptually the boundary transfers are much more difficult, since there is generally no regular relationship between the (deemed) boundary points on successive levels of discretization. Figure 6.51 shows the computational geometry of a two level structure. The finer mesh is displayed with squares whereas the coarser mesh is displayed with circles. We apply boundary conditions at the open circles and squares.

There is a problem in using straightforward multigrid ideas on such domains since there is no natural two-to-one relationship between the coarse and fine points. However, if we view the boundary points as actually coinciding with the physical boundary, we can think of the boundary transfers in terms of prolongations and restrictions along the θ direction. The transfers become easy to formulate in terms of appropriate interpolations (generally linear) in the θ direction. For example, the restriction operator is:

$$
\begin{aligned}
I_h^H u_b^h(l) = u_b^H(L) &= u_b^h(l_2)\,\frac{\theta^h(l_1)-\theta^H(L)}{\theta^h(l_1)-\theta^h(l_2)} \\
&+ u_b^h(l_1)\,\frac{\theta^h(l_2)-\theta^H(L)}{\theta^h(l_1)-\theta^h(l_2)} \qquad (6.82)
\end{aligned}
$$

where l_1 and l_2 are the indices of the two fine-grid points which are closest to the coarse grid point with index L. (Note that here the superscripts on the θ array indicate with which grid the array is associated.)

Prolongation is defined in an analogous fashion: the roles of the fine and coarse grids are simply reversed:

$$
I_H^h u_b^H(L) = u_b^h(l) = u_b^H(L_2)\,\frac{\theta^H(L_1)-\theta^h(l)}{\theta^H(L_1)-\theta^H(L_2)}
$$

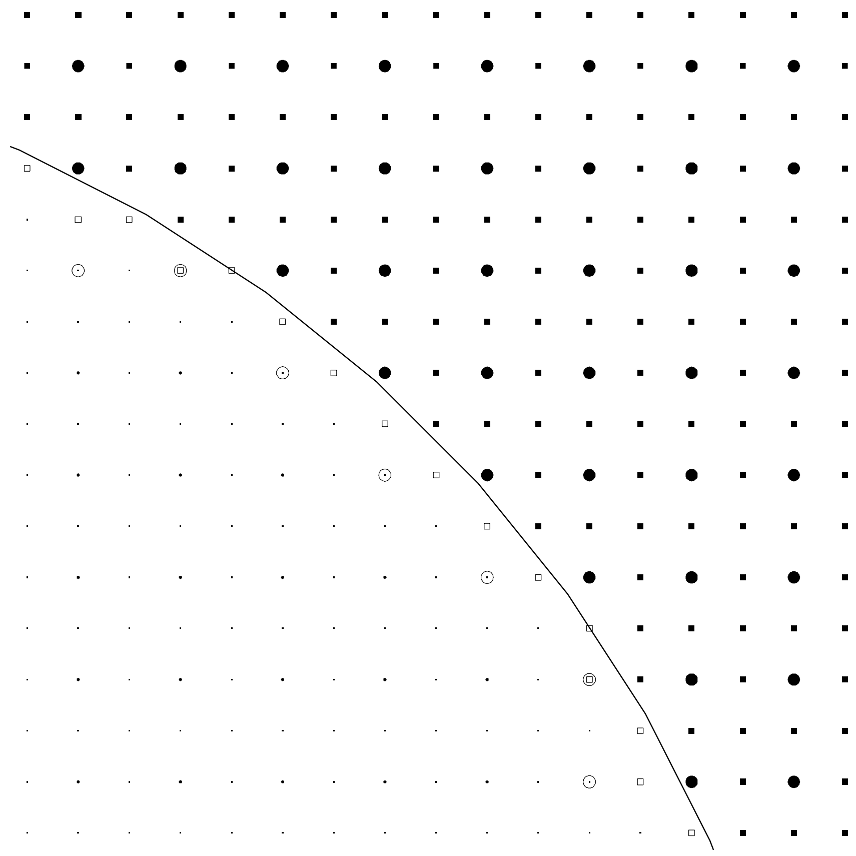
$$+ \quad u_b^H(L_1)\,\frac{\theta^H(L_2) - \theta^h(l)}{\theta^H(L_1) - \theta^H(L_2)} \tag{6.83}$$

Here, L_1 and L_2 are the indices of the two coarse-grid points which lie nearest to the fine grid point with index l. We do not claim that these operators define the best transfers for our model problem but we have found them to be adequate for our purposes.

Again, to minimize computational time we precompute the coefficients needed for the transfer operators.

Figure 6.51: A close up view of the geometry of two levels.

The finer level is shown with squares, and the coarser level is referenced with circles. The points which are not filled in are the ones where the boundary equation will be applied. Notice that the 2 grids do not align with each other.



6.7.4 Deferred correction for 2D code

As usual we will generally use fourth-order deferred correction. To compute an appropriate fourth order stencil, we use the routines described in section(3.2.4).

We have found that by using deferred correction to fourth order, we sometimes slow the convergence rate down by almost 50%. We must spend more time to fix this potential problem.

6.7.5 The algorithm

Figures (6.52) and (6.53) define the complete algorithm whose components have been described above. These routines use stacks to implement general μ cycles. To learn about stacks, we recommend the literature by Knuth[32]. The routine **`pushlevel`**, pushes pending commands, such as **`relax`**, plus level information onto a single stack which is used on all levels. **`querylevel`** checks the top of the stack to see which command should be executed, and at which level. Once a command has been completed, **`poplevel()`** is invoked to delete the corresponding entry from the top of the stack.

The **`upstroke`** command determines τ_h^H and then adds it to f^H and g^H. Similarly the **`downstroke`** command determines a prolonged correction after a coarse grid correction and adds it to u^h. The deferred correction scheme is implemented in the **`upstroke`** routine and makes use of the **`EVAL_MATRIX`** routine to compute the high order interpolant described in chapter 3. The variable **`nwcyc`** is a parameter which encodes which type of μ-cycle is to be used. For example, V- and W-cycles are performed for the cases **`nwcyc`** $= 1$ and **`nwcyc`** $= 2$, respectively. Another aspect of this algorithm which distinguishes it from our previous multigrid algorithms is that we differentiate between the number of times we will smooth on the finest levels, (**`pref`** and **`pstf`**), and on all other levels, (**`pre`** and **`pst`**). Finally, **`bsweeps`** defines the number of times we relax the boundary per interior sweep.

Figure 6.52: The 2D MG algorithm

```
PROCEDURE MG2D_HOLE( l_max , ncyc , nwcyc, order , pref, pstf ,
                        pre , pst , bsweeps )
   Set up memory initially on all levels
   DOlev = l_coarse , l_max    Loop over all the levels
      GET_CHAR( nx(lev) , ny(lev) , char,n_P)
      Loop over all the points where char=2 and get the stencil coefficients.
      n_P(lev) = 0 ,    Set the number of points on the boundary=0
      DOk = 1 , ny(lev)
         DOj = 1 , nx(lev)
            IF( char(j,k) = 2 ) THEN
               n_P(lev) = n_P(lev) + 1
               DETERMINE_STENCIL( )
               ı(n_P(lev)) = j
               ȷ(n_P(lev)) = k
               GET_LISTS( )
               GET_NORMALS( )
               GET_BOUNDARY_COEFF( )
            END IF
         END DO
      END DO
      Sort the precomputed lists in ascending order in θ
      Now compute the deferred correction LUD matrix
      DEF_MATRIX( )
   END DO
   Now compute boundary transfer coefficient
   DOlev = l_coarse , l_max    Loop over all the levels
      GET_RESTRICTION( θ(lev − 1),θ(lev), θ_1r , θ_2r)
      GET_PROLONGATION( θ(lev + 1),θ(lev), θ_1p ,θ_2p)
   END DO
   Now start to do a FMG cycle
   Now solve the system on the coarsest level
    SOLVE(L^H u^H = f^H , and B_1^H u^H = g^H
   DOlev = l_coarse + 1, l_max
   Use linear prolongation to prolong u^H to u^h
   PROLONG( u^h := II_H^h(u^H) )
      CYCLE( )
   END DO
END PROCEDURE
```

Figure 6.53: The μ cycle algorithm, with the implementation of stacks

```
PROCEDURE CYCLE( )
this routine uses stacks which determine the commands to issue
   l := l_max
   pushlevel(l,ncyc,pre,pst,l_max)
   START:  continue
      s := querylevel(l,swptype)
      if ( s= 0) goto DONE
      case ( swptype=pre)
         r := relax(l)
         poplevel()
      case ( swptype=pst)
         r := relax(l)
         poplevel()
      case ( swptype=solve)
         call solve(l)
         poplevel()
      case ( swptype=upstroke)
         call upstroke(l)
         poplevel()
         l = l - 1
         pushlevel(l,nwcyc,pre,pst,l_max)
      case ( swptype=downstroke)
         call downstroke(l)
         poplevel()
         l = l + 1
      goto START
   DONE: continue
END PROCEDURE
```

6.7.6 Results

We first present convergence rates for a variety of tests which clearly show that we can effectively use the multigrid algorithm with non-trivial (*i.e.* non-conforming) boundaries. We then perform additional tests to demonstrate convergence and the use of deferred correction.

Our first test applies the multigrid algorithm described above to a 5-level approximation of equation (6.77), with $c_x = c_y = 0$, $a = 1$. The finest level has roughly 257×257 points and we use a variety of μ cycles. Table 6.34 summarizes the results of this first set of experiments in which we examine the convergence rate as a function of **`nwcyc`**, **`pref`**, **`pstf`**, **`pre`**, **`pst`**,and **`bsweeps`**. For each run we were able to get the ℓ_1 norm of the residuals below 1.0e-06. The best convergence rate, which is highlighted with boldface in the table, is approximately -2.0e-01, which means that for every 5 relaxation sweeps the ℓ_1 norm of the residuals are reduced by one order of magnitude. This is slightly slower than typical model problems which have been discussed in the literature[9].[5]

We then solve a 6-level system and perform the same parameter-space survey. Table 6.35 shows the convergence rate, ρ, for this series of tests. We observe only a slight decrease in the convergence rate from the 5-level runs. We also see that the optimal parameter settings are the same in the two cases.

Fixing the input parameters to their empirically determined values, we now investigate if the convergence rates remain approximately constant as we vary $l_{\max}$. Table 6.36 shows that the convergence rate *does* remain approximately constant. This clearly shows that the code converges in $O(1)$ cycles (or, equivalently, $O(1)$ work units).

We now examine the impact of fourth-order deferred correction on the convergence rate (Table 6.37). We see that for the 5-level runs summarized in the table, the best input parameters are just slightly different from the uncorrected case. Relative to the uncorrected runs, the convergence rate goes down by a factor of $2 \sim 3$, but remains relatively constant. These runs show that the residual norm is reduced by one order of magnitude for every 12 or so sweeps. This may be seem like poor performance by traditional multigrid standards, and there is clearly room for improvement. However, as we now discuss, the performance of the code with respect to the reduction of the *error* in the difference solution is actually quite good.

In Figure 6.54 we schematically depict a 2 level system where we also show some boundary reference points. In order to fully test to see if the results are extrapolatable, we interpolate the solutions to fourth order, onto reference points, which are uniformly displaced around the boundary, as shown in the figure.

[5] Such applications generally have convergence rates around -3.3e-01.

Table 6.34: Results from the 2D FMG FAS algorithm with $a = 1$, when we vary the input parameters

Deferred correction not used, $x_{\max} = y_{\max} = 2.4$, 5 level system

`nwcyc`	`pref`	`pstf`	`pre`	`pst`	`bsweeps`	ρ
1	1	1	1	1	1	-2.0(-1)
1	**2**	**1**	**1**	**1**	**1**	**-2.0(-1)**
1	3	1	1	1	1	-2.0(-1)
1	4	1	1	1	1	-1.9(-1)
1	1	2	1	1	1	-1.6(-1)
1	1	3	1	1	1	-1.3(-1)
1	1	4	1	1	1	-1.0(-1)
1	1	1	2	1	1	-1.8(-1)
1	1	1	3	1	1	-1.5(-1)
1	1	1	4	1	1	-1.4(-1)
1	1	1	1	2	1	-1.7(-1)
1	1	1	1	3	1	-1.5(-1)
1	1	1	1	4	1	-1.3(-1)
1	2	1	1	1	2	-2.0(-1)
2	2	1	1	1	2	-1.7(-1)

Figure 6.55 shows the errors on several levels computed from runs with and without deferred correction. The coarse-grid and fine-grid results are shown with large and small symbols respectively. Squares are used for the uncorrected run and open circles for the corrected run. The solid circles show errors in the Richardson extrapolated results produced by application of equation (4.40),

$$\frac{4}{3}\,u^h - \frac{1}{3}\,u^{2h} = u + O\left(h^4\right), \tag{6.84}$$

to the two levels of corrected quantities. The graph clearly hows that deferred correction (1) reduces the overall error in the "raw" solution, and (2) produces values which are extrapolatable. In particular, these results show that with about twice the amount of work the error in the fine grid values (after extrapolation) is reduced by almost three orders of magnitude!

Thus, although more study is needed in order to improve the overall convergence rate, we have clearly demonstrated that we can obtain highly accurate results on non-trivial domains (and using non-conforming coordinates), using multigrid as our elliptic solver.

Table 6.35: Results from the 2D FMG FAS algorithm with $a = 1$, and no deferred correctiond

$x_{\max} = y_{\max} = 2.4$, 6 level system

nwcyc	pref	pstf	pre	pst	bsweeps	ρ
1	1	1	1	1	1	-1.6(-1)
1	**2**	**1**	**1**	**1**	**1**	**-1.8(-1)**
1	3	1	1	1	1	-1.7(-1)
1	4	1	1	1	1	-1.6(-1)
1	1	2	1	1	1	-1.4(-1)
1	1	3	1	1	1	-1.1(-1)
1	1	4	1	1	1	-8.9(-2)
1	1	1	2	1	1	-1.4(-1)
1	1	1	3	1	1	-1.3(-1)
1	1	1	4	1	1	-1.1(-1)
1	1	1	1	2	1	-1.4(-1)
1	1	1	1	3	1	-1.3(-1)
1	1	1	1	4	1	-1.1(-1)
1	2	1	1	1	2	-1.7(-1)
2	2	1	1	1	2	-1.4(-1)

Table 6.36: Results from the 2D FMG FAS algorithm with $a = 1$

Deferred correction not used, $x_{\max} = y_{\max} = 2.4$

$l_{\max}$	ρ
2	-2.9(-1)
3	-2.9(-1)
4	-2.0(-1)
5	-2.0(-1)
6	-1.8(-1)

Table 6.37: Results from the 2D FMG FAS algorithm with $a = 1$

Deferred correction not used, $x_{\max} = y_{\max} = 2.4$, 5 level system

nwcyc	pref	pstf	pre	pst	bsweeps	ρ
1	1	1	1	1	1	-5.9(-2)
1	2	1	1	1	1	-4.9(-2)
1	3	1	1	1	1	-4.8(-2)
1	4	1	1	1	1	-4.8(-2)
1	1	2	1	1	1	-7.2(-2)
1	1	3	1	1	1	-5.2(-2)
1	1	4	1	1	1	-4.1(-2)
1	1	1	2	1	1	-5.4(-2)
1	1	1	3	1	1	-4.7(-2)
1	1	1	4	1	1	-4.1(-2)
1	1	1	1	2	1	-5.3(-2)
1	1	1	1	3	1	-4.7(-2)
1	1	1	1	4	1	-4.2(-2)
1	1	1	1	1	2	-6.5(-2)
1	1	1	1	1	3	-6.5(-2)
1	**1**	**2**	**1**	**1**	**2**	**-7.2(-2)**

Table 6.38: Results from the 2D FMG FAS algorithm with $a = 1$ using deferred correction to fourth order

$x_{\max} = y_{\max} = 2.4$

$l_{\max}$	ρ
3	-1.0(-1)
4	-9.1(-2)
5	-7.2(-2)
6	-8.2(-2)

Figure 6.54: 2D domain showing 2 levels, and the reference points.

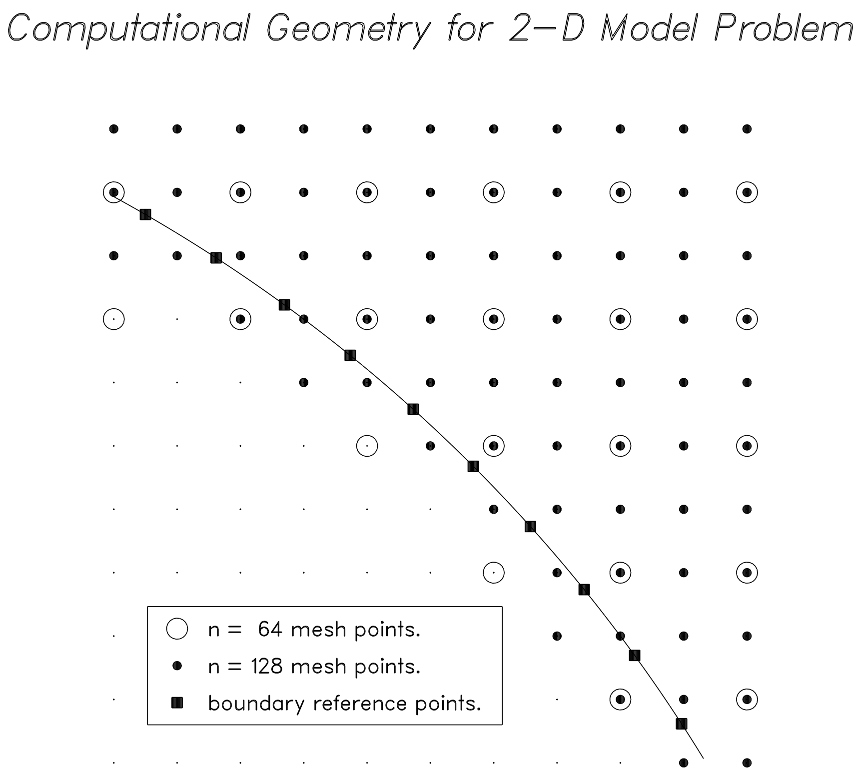
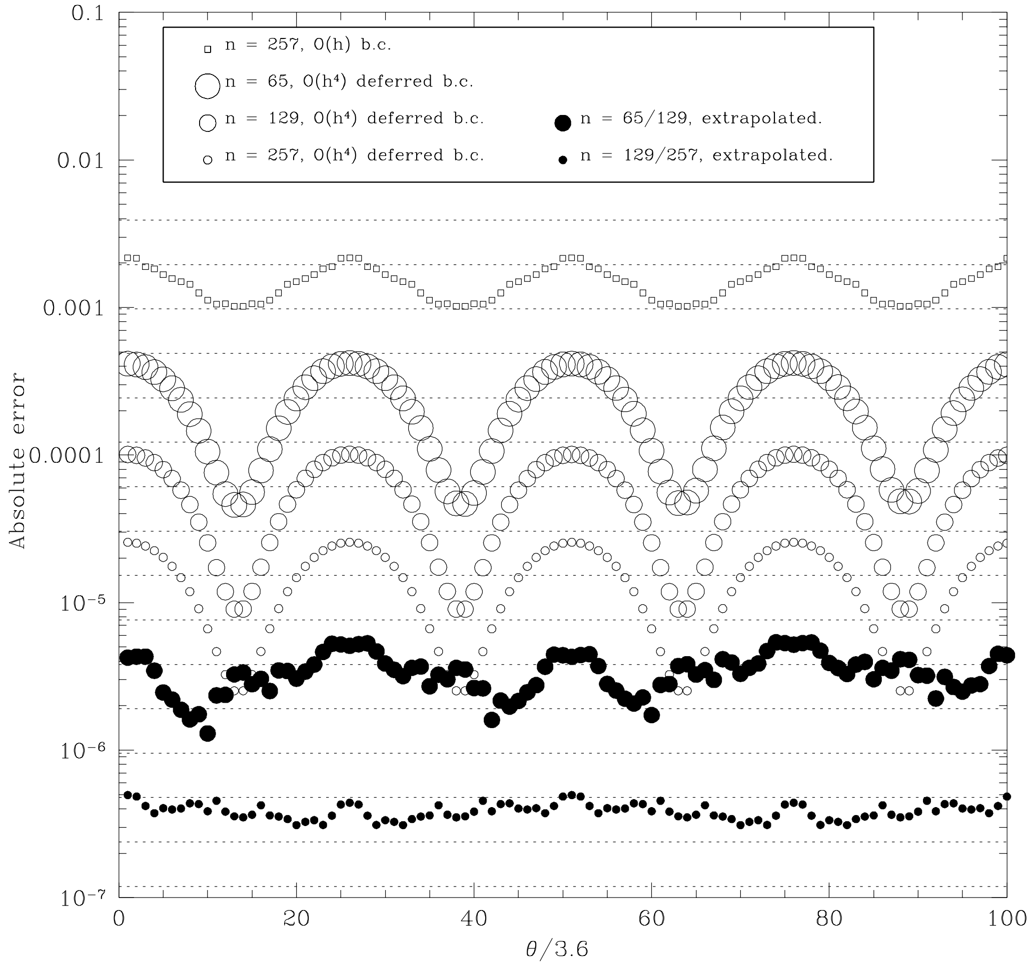


Figure 6.55: The absolute value of the errors on the interpolated points.



Chapter 7

Conclusion

In this chapter we summarize the results and conclusions of this dissertation. We also briefly outline possible future research based on the work we have presented here.

Our goal was to design a computer code which would be fast and accurate for solving the Initial Value Problem (IVP) in three dimensions using Cartesian coordinates. In Chapter 2 we saw how the Einstein equations can be considered as evolution equations and initial value constraint equations. Later we saw the initial value constraint equations were elliptic equations with Robin boundary conditions. Now the goal of this dissertation was to design an algorithm to solve this elliptic PDE, to fourth order accuracy, using the MLAT techniques, and Richardson extrapolation.

We have yet to implement a code which uses all of the tools we presented here to solve the IVP. We did, however, generate reasonably accurate data sets using **`genpsi`**, shown in Chapter 5, which was originally developed as a test-bed for the design of a multigrid code which would solve the same equations. We showed in Chapter 5 that although we were only able to get first order accurate solutions, our solutions were still accurate enough to represent the initial data. We also presented results from another test code, which was able to produce fourth order extrapolatable results for a model problem. Thus, the two goals we set for ourselves was first to design a code that would be extrapolatable such that the solution would be accurate to fourth order, and second, to design a code that would take $O(1)$ work units. Since we have non-boundary conforming meshes, both of these goals were difficult to achieve.

The first problem that we encountered was in implementing both the inner and outer boundary conditions. Here we found that in order to investigate the problem in detail, it was advantageous to consider one dimensional model problems, which will hopefully generalize for the multidimensional cases. The next problem we encountered was in implementing a Berger and Oliger[2] adaptive scheme. We used Choptuik's memory storage scheme and tree links to design the data structures that we will use in our final code. We also saw the problem of using AMR and attempting to extrapolate. We saw that straightforward applications of the basic MLAT algorithm[9] the solution will only be first order accurate. Therefore, we empirically found a scheme which produced extrapolatable results. This scheme smoothed the relative truncation error estimate across the interfaces. We found that by using this scheme we could always achieve fourth order results from Richardson extrapolating the solutions. Another problem we saw was that grids became uncontained from their parents, since the finer grids would be closer to the actual boundaries. Since the points inside of the holes are not defined in the computational domain, we saw that the finer grids approached the boundaries closer than the coarse grids. We empirically determined transfer operators which allowed the convergence rate to remain constant. We also applied the blended boundary condition, which may prove to be useful for multi-dimensional codes.

Our last problem in designing such a code was to implement the multigrid algorithm for non-aligned grids. We saw earlier that, by determining the proper boundary smoothing operators and transfers, we were able to develop a multigrid code which could be extrapolatable via deferred correction. We saw that deferred correction is an extremely powerful tool, which we plan to use in the future for all of our elliptic multigrid algorithms .

The work for the future is to incorporate the ideas in this dissertation into a general 3D multigrid code. At this point we would be able to determine the solution of the IVP up to the same order of accuracy as the boundary fitted Čadež coordinate multigrid code developed by Cook. Our plans are then to apply our knowledge in handling the boundaries to the evolution code. We also plan on implementing general 3D multigrid codes which could be used to determine the initial conditions for multiple black holes. Since other elliptic equations arise in the evolution equations, we are also planning to develop MLAT solvers for these equations

Appendix

Appendix A

1D parallel multigrid implementation

Here we explore the parallelization of a one dimensional multi-grid algorithm on an Intel IPSC hypercube with 32 processors[38]. We find that our algorithm is scalable[1], and is implemented much like that of the serial algorithm, shown in chapter(4). Our technique is to use all the available processors on levels which have over 129 points. When the total number of points in the domain is less than this number, we switch to letting the entire problem be solved on all processors identically. We find that if we do not use this technique, the algorithm becomes non-scalable, and in fact, the time required to solve the system begins to increase as the number of processors is increased. We feel that the techniques utilized in our algorithm can be directly applied to the higher dimensional cases. This code was written by this author and Reid Guenther[29] for a class by R. Van de Geign.

Since multigrid is the best method for solving elliptic systems, it is quite clear that one needs to establish a parallel algorithm for this method, since we will in general be handling very large systems of equations. In this appendix we only look at one dimensional methods, since we feel that the knowledge gained here will go into the much larger effort of designing a three dimensional method.

Much work has been done already, but mostly in higher dimensions. For example , one of the NAS benchmarks[2] is for a two dimensional multigrid code. It is quite clear that from all of the NAS benchmarks, that multigrid is the fastest method to solve elliptic equations, (even though this method will never achieve the FLOPS of some of the other methods).

A.1 Target architectures

We implement this algorithm on a hypercube consisting of 2^D $(D = 0 \cdots 5)$. independent processors, each with its own local memory. There is no shared memory available- the

[1] A scalable algorithm is one which time $\sim p^{-1}$, where p is the number of processors.
[2] A common benchmark program used today to rate the performance of computers

processors cooperate by message passing. Messages are passed over the interconnection network which is a hypercube in a space of dimension D. Processors are located at the vertices of the D-dimensional hypercube and adjacent vertices of the cube are connected by a communication channel along the corresponding edge[38].

Our parallel implementation of the one dimensional multigrid method assumes the use of a one dimensional mesh with 2^D nodes, where each node is a separate processor. We also assume that there are bidirectional links between nodes and worm-hole (cut-through) routing. Furthermore, since we are performing message passing only between nearest neighbors on the hypercube, we assume it is possible to model the time required for sending a message of length n bytes between any two nodes by

$$\alpha + n\beta$$

where α equals the latency of the network, and β is the time per item, in the absence of network conflicts[52]. Communication is single ported, meaning a processor can only send to, or receive from, one other node at a given time. When two messages traverse the same physical link, we assume they time-share the bandwidth of that link.

A.1.1 Description of the basic parallel algorithm

There are only two basic changes to the serial code shown in Figure 4.11. One is that the physical 1D mesh is broken up onto the computational ring of nodes with adjacent parts of the mesh lying on adjacent nodes. Now, for the problem we are solving, a given physical point will need information from its left and right neighbors. Thus the left and right most mesh points on a node will need information from their physical neighbor on the adjacent node. So we use shadow points to communicate between nodes:

In Figure A.1, we show part of the physical mesh mapped onto two adjacent physical computational nodes. Point x is a "shadow point" for node m while point $x-1$ is a shadow point for node $m+1$. Node m will use the information inherent at point x to update information at point $x-1$ but not update point x, while the opposite is true on node $m+1$. So when the information at $x-1$ and x are updated by their respective nodes, this data is communicated to the adjacent node and stored at the shadow points on each node. So given that we have 2^l+1 mesh points on 2^D nodes, we divide the mesh in this manner in the figure below, with the left and right most mesh point on a node being a shadow point for that node except for mesh points 1 and 2^l+1 which are physical boundary points.

The second difference between the serial and parallel codes involves communication versus computation time. In the V-cycles of the parallel multigrid code computations are being performed at different levels l which involve $O\left(2^{l-D}\right)$ computations while communication between nodes remains constant at each level in the V-cycle. Thus, there will be a level below which the communication time will be longer than the computation time. Around this level, we have the parallel code finish the lower part of the V-cycle in serial fashion. This is probably best understood by going through the pseudo-code shown in figures (A.2, A.3).

We see in the algorithms that in going up to the coarsest level in the V-cycle routine, there is a level $l_{\rm mid}$ at which all nodes begin solving the problem in serial fashion. Thus no communication is needed between nodes for these levels. Then cycling from the coarse grid up to the finest grid, the routine goes back to parallel. Since $l_{\rm mid}$ is an adjustable parameter we can experiment to see which is the best level to implement the transition from a parallel to a serial code and vice versa.

The other difference in this code and the serial code is the updating of the shadow points. As one sees from the pseudo-code this is done only in the smoothing and restriction routines. Thus the differences between our parallel 1-D multigrid code and the serial code are cosmetically simple but produce some very encouraging results which will be discussed in Section A.2.

A.1.2 Discussion of improvements that yield high performance

Since our parallel code utilizes the same multigrid routines as the serial code except for the updating of shadow points in the smooth and restriction routines, it is highly vectorized. The updating of shadow points uses the basic `csend` and `crecv` routines on the iPSC/860 which we assume are efficient routines.(We also implemented level 4 optimization on the iPSC and received a 30% speed up.)

One problem with our code is the global collect we use to transfer the problem from parallel mode to serial mode. On the hypercube, the ring is embedded using a Gray code while the global collect routine is for a hypercube. So two more `csend` and `crecv` calls are needed to have a global collect for a ring. We used the iCC[51] call for the global collect, but a collect optimized for a ring embedded on the hypercube should be much faster. This is not a serious problem as the collect occurs only once in the cycle.

A.1.3 Estimated run time and scalability

The time for the serial multigrid code to perform a V-cycle on the problem:

$$\frac{\partial^2 u(x)}{\partial x^2} = f(x), \quad 0 < x < 1 \tag{A.1}$$

with Dirichlet boundary conditions on a mesh with $2^{l_{\max}} + 1$ points is about:

$$T_s = (C_{DN} + C_{UP})\,\gamma \sum_{n=1}^{l_{\max}} 2^n \simeq (C_{DN} + C_{UP})\,\gamma 2^{l_{\max}+1} \tag{A.2}$$

where γ is the time for an add or multiply, C_{DN} is the number of calculations in the down part of a V-cycle at each level and C_{UP} is defined similarly. This number is calculated simply by examining the code for how many adds and multiplies must be performed. The time for the parallel code to perform one cycle of the same problem on 2^D nodes is:

$$\begin{aligned} T_p = & (\texttt{pre}+3)\,(l_{\max}-l_{\mathrm{mid}})\,M_D\,(\alpha+8\beta) \\ & + \sum_{n=l_{\mathrm{mid}}+1}^{l_{\max}} C_{DN}2^{n-D}\gamma + \sum_{n=1}^{l_{\mathrm{mid}}} C_{DN}2^{n}\gamma \\ & + (\texttt{pst}+1)\,(l_{\max}-l_{\mathrm{mid}}+1)\,M_D\,(\alpha+8\beta) \\ & + \sum_{m=l_{\mathrm{mid}}}^{l_{\max}} C_{UP}2^{n-D}\gamma + \sum_{n=1}^{l_{\mathrm{mid}}-1} C_{UP}2^{n}\gamma \end{aligned} \tag{A.3}$$

where $l_{\max}$, γ, C_{up} and C_{DN} are the same as above, α and β are defined in Section A.1, l_{mid} is the level at which the problem switches from parallel to serial mode, M_D is the number of message passing startups a node must perform in the red-black smoothing routine and is 2 for $D=1$, because each node only has one shadow point, and 4 for $D>1$, *pre* is the number of pre-sweeps of the smoother and *pst* is the number of post-sweeps of the smoother. Equation (A.3) is more complicated than (A.2) because it involves message passing and a transition between running in parallel and serially. The first line of the equation gives the time for the message passing in cycling down the V. The second line is the time split between running in parallel from level $l_{\max}$ to l_{mid} and serially down to l_{mid}. The third and fourth lines are similar to the first two but involve going up the V-cycle.

The best way to compare how well a parallel code performs compared to a serial code is to calculate the efficiency, which is just the time it takes serial code to run divided by the time it takes the parallel code to run divided by the number of nodes on which the parallel code runs. Thus, due to the overhead of message passing, the efficiency will lie between 0 and 1. For $\texttt{pre}=\texttt{pst}=2$,

$$\frac{\alpha+8\beta}{\gamma}\simeq 1000$$

, $M_D=4$, $C_{DN}=55$ and $C_{UP}=37$, the efficiency of the code is:

$$\begin{aligned} E & \equiv \frac{T_s}{2^D T_p} \\ & \simeq \frac{1}{1+\frac{140}{184}\left(229\,(l_{\max}-l_{\mathrm{mid}})+2^{l_{\mathrm{mid}}}\right)2^{D-l_{\max}}} \end{aligned} \tag{A.4}$$

This calculation appears to be very encouraging since as $l_{\max}$ increases, the efficiency goes to 1. The term in parenthesis shows that l_{mid} should remain constant as $l_{\max}$ grows to maintain efficiency since $2^{l_{\mathrm{mid}}}$ grows much faster than $(l_{\max}-l_{\mathrm{mid}})$ when increasing l_{mid} and $l_{\max}$ by the same increment. Another good feature is that to maintain efficiency when increasing the number of nodes the problem size should grow proportionally. Thus the code should be scalable.

A.2 Experimental results

Given the time constraints on this project, only a rudimentary analysis of the code has been performed. Our first result was a comparison of run time versus $l_{\rm mid}$ for a given problem ($l_{\rm max} = 14$, `pre` = `pst` = 2 on 8 nodes). As one sees in Table A.1, the times have a minimum for $l_{\rm mid} = 7$. This is to be expected since a conflict occurs between when communication time overtakes computation time and how much extra computation must be done in serial fashion. So one would expect to find a level $l_{\rm mid}$ at which the run time is minimized.

Table A.2 gives an indication of the scalability and efficiency of this code for ($l_{\rm mid} = 7$) at various maximum levels and cube dimensions. Our results of the previous section predicted that $l_{\rm max}$ should scale proportionally to D to maintain scalability. As one sees by looking diagonally down the chart, the problem does not scale very well when $l_{\rm max}$ is small, but as $l_{\rm max}$ increases, the experimental results comply with the calculated results given in the above section. One thing to note is the jump in times as one moves from $D = 1$ to $D = 2$ and increment $l_{\rm max}$ by 1 for the lower $l_{\rm max}$s. There is a consistent decrease that goes down by less than a factor of 2. This is because the number of communications is doubling when one moves form 2 nodes to 4 due to each node having only one shadow point for 2 nodes while the interior nodes have 2 shadow points for 4 nodes and more. Table A.2 also gives an indication of the efficiency of this code. When $l_{\rm max}$ is small, the efficiency is not very good but as $l_{\rm max}$ increases, one sees that $T_p \simeq 2^D T_s$ as predicted above.

A.3 Conclusion

Our results are very encouraging. We have a parallel one dimensional multigrid code that is scalable for large problems. Thus we will try to implement the knowledge gained here into a multi-dimensional parallel multi-grid code that will be used directly in our research. This should not be as straight forward as writing the one dimensional code since the message passing to shadow points will no longer be a constant. We expect that we will have to have multiple levels in our cycling at which we reduce the number of nodes running different parts of the problem to keep the amount of message passing below the amount of computation on a given level.

Table A.1: No optimization, cube dimension =3

Variation of l_{mid} for l_{max} = 14

l_{mid}	time (s)
4	9.2(-2)
5	8.9(-2)
6	8.6(-2)
7	8.4(-2)
8	8.5(-2)
9	8.8(-2)
10	9.7(-2)
11	1.20(-1)
12	1.7(-1)

Table A.2: No optimization, l_{mid} = 7

Scalability of 1D Multigrid Code normalized for smallest time = 1.0, Normalization factor = 7.626e-03^{-1}

l	0	1	2	3	4	5
8	1	1.68	2.06	2.23	2.03	
9	1.93	2.49	2.84	2.85	2.53	
10	3.87	3.77	3.78	3.63	3.24	
11	7.75	6.12	5.27	4.54	3.95	
12	15.52	10.35	7.74	6.09	5.04	4.60
13	30.97	18.42	11.95	8.38	6.51	5.83
14	61.95	34.34	20.23	12.85	8.92	7.06
15	123.83	65.64	36.21	21.05	13.22	9.17
16	247.44	128.02	67.53	37.00	21.39	13.81
17	494.88*	252.29	129.96	68.32	37.20	21.94
18	989.76*		253.86	130.74	68.25	37.63
19	1979.52*			254.92	131.05	68.35
20	3959.04*				252.56	131.85
21	7918.08*					253.08

node 0 gets mesh points	2^{D-l} +1
node 1 gets mesh points	$2^{D-l} - 2^{D-l+1}$ +1
.	.
.	.
.	.
node 2^{D-1} gets mesh points	$2^{l-1} - 2^{l}$ +1

Figure A.1: A schematic of two nodes, sharing the "shadow" points

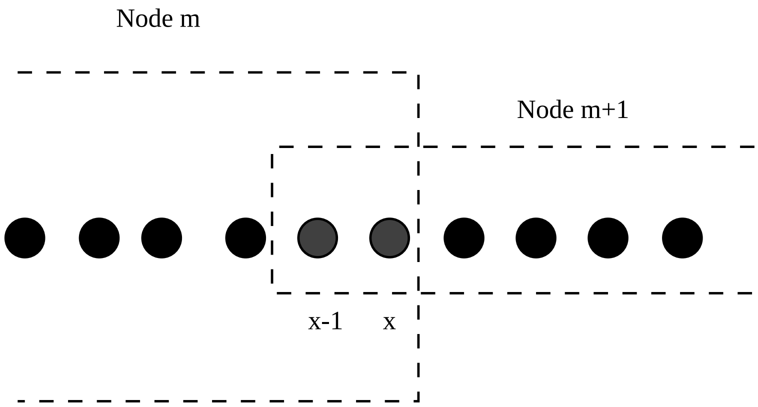


Figure A.2: The parallel V-cycle algorithm

PROCEDURE VCycle(cycle, $l_{\max}$, l_{mid}, pre, pst)
    Cycle up to the coarsest level
    DO l = $l_{\max}$, $l_{\text{coarse}} + 1$
        IF $(l = l_{\text{mid}})$ THEN
            Begin serial processing on all nodes
        IF $(l \neq l_{\max})$ OR $(\texttt{cycle} = 1)$ THEN
            DO p = 1 , pre
                SMOOTH(l)
            END DO
        END IF
        Now get a truncation error estimate, and update f^H
    END DO
    Now solve the system on the coarsest level
    SOLVE(l_{coarse})
    DO l = l_{coarse} , $l_{\max} - 1$
        IF $(l = l_{\text{mid}})$ THEN
            Begin parallel processing
        Determine the correction
        $u^h := u^h + I_H^h \left(u^H - I_h^H u^h\right)$
        Now smooth out u^h
        DO q = 1 , pst
            SMOOTH(l)
        END DO
    END DO
END PROCEDURE

Figure A.3: The parallel smoothing algorithm

PROCEDURE SMOOTH(l)
    First smooth the even points
    DOj = 2 , n-1 , 2
        $u[j] := 1/2\left(u[j+1] + u[j-1] - h^2 f[j]\right)$
        update shadow points
    END DO
    Now smooth the odd points
    DOj = 3 , n-1 , 2
        $u[j] := 1/2\left(u[j+1] + u[i-1] - h^2 f[j]\right)$
        update shadow points
    END DO
END PROCEDURE

Bibliography

[1] R. Arnowitt, S. Deser, and C. W. Misner, in *Gravitation- An Introduction to Current Research* edited by L. Witten (Wiley, New York, 1962).

[2] M.J. Berger, and J. Oliger, J. Comput. Phys. **53** 484 (1984).

[3] M.J. Berger, and I. Rigoutsos, IEEE Trans. on systems, man, and cybernetics **21** 1278 (1991).

[4] J. M. Bowen, General Relativity and Gravitation **11** 227 (1979).

[5] J.M. Bowen, Ph.D. dissertation, University of North Carolina at Chapel Hill, (1979).

[6] J. M. Bowen, General Relativity and Gravitation **14** 1183 (1982).

[7] J. M. Bowen, and J. W. York, Jr., Phys. Rev. D **21** 2047 (1980).

[8] A. Brandt, Math. Comput. **31**,333 (1977).

[9] A. Brandt, in *Lecture Notes in Mathematics: Multi Grid Methods*, Vol. 960, W. Hackbusch and U. Trottenberg, eds. (Springer Verlag, New York).

[10] W. Briggs, in *A Multigrid Tutorial*, (Society for Industrial and Applied Mathematics) Philadelphia, 1987.

[11] M.W. Choptuik, Phys. Rev. D. **44** 3124 (1991).

[12] M. W. Choptuik, and W. G. Unruh, General Relativity and Gravitation **18** 813 (1986).

[13] M. W. Choptuik, D. S. Goldwirth, and T. Piran, Classical and Quantum Gravity **9** 721 (1992).

[14] M.W. Choptuik, Personal communication.

[15] M.W. Choptuik, M. Sc. Thesis, University of British Columbia, (1982).

[16] M. W. Choptuik, in ad *An Implementation of the Berger-Oliger Mesh Refinement Algorithm for the Wave Equation in Spherical Symmetry* (1994).

[17] B. Choquet and J. W. York, Jr., in Einstein Centenary Volume I edited by A. Held (Plenum, New York, 1980).

[18] G. B. Cook, in *Frontiers in Numerical Relativity* edited by C. R. Evans, L.S. Finn and D. W. Hobill (Cambridge University Press, England, 1989).

[19] G. B. Cook, Phys. Rev. D **44** 2983 (1991).

[20] G.B. Cook, M.W. Choptuik, M. R. Dubal, S.A. Klasky, R. A. Matzner,and S.R. Oliveria, Phys. Rev. D **47**, 1471 (1993).

[21] G.B. Cook and J. W. York Jr., Phys. Rev. D **41** 1077 (1990).

[22] G. B. Cook, Ph.D. dissertation, North Carolina at Chapel Hill, (1990).

[23] G. B. Cook and A. M. Abrahams, Phys. Rev. D. **46** ,702 (1992).

[24] A. Einstein and N. Rosen, Phys. Rev.**78** 73 (1935).

[25] K. R. Eppley, Ph.D. dissertation, Princeton University, (1975).

[26] C. R. Evans, L. S. Finn, and D. W. Hobill, in *Frontiers in Numerical Relativity* (Cambridge University Press, England, 1989).

[27] J.A. George, SIAM Journal of Numerical Analysis **10** 345 (1973).

[28] J.A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems.* (Prentice-Hall, N.J., 1981).

[29] R. L Guenther and S. A. Klasky, *A Parallel Implementation of Multi-Grid in one dimension* (Site report, 1994).

[30] S. Hahn, and R. Lindquist, Ann. Phys., **29** 304 (1964).

[31] M. Huq Personal communication.

[32] D.E. Knuth, *The art of computer programming* (Addison Wesley, Mass., 1968).

[33] A.D. Kulkarnia, L.C. Shepley, and J. W. York, Jr., Phys. Lett. A **96** 228 (1983).

[34] A.D. Kulkarni, Ph.D. dissertation, University of Texas at Austin, (1984).

[35] A. Lichnerowicz, J. Math. **23** 37 (1944).

[36] C.W. Misner, K. S. Thorne, and J.A. Wheeler, *Gravitation* (W. H. Freeman, San Francisco, 1973).

[37] R. A. Matzner, Personal Code .

[38] O. A. McBryan and E. F. Van de Velde, in *Lecture Notes in Mathematics: Multi Grid Methods II*, Vol 1228, W. Hackbusch and U. Trottenberg, eds. (Springer Verlag, New York,1985).

[39] C. W. Misner, Ann. Phys. (N.Y.) **24**, 102 (1963).

[40] A. R. Mitchell, and D.F. Griffiths, in *The finite difference method in partial differential equations* (Wiley, Chichester, 1980).

[41] W. H. Press, b. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing* (Cambridge University Press, Cambridge, England, 1986).

[42] U. R̈uede, Personal communication.

[43] J. D. Rauber in *Dynamical Spacetimes and Numerical Relativity* edited by J.M. Centrella (Cambridge University Press, England, 1986).

[44] L. F. Richardson, Philos. Trans. R. Soc. **210** 307 (1910).

[45] S. McCormick and U. Rüde, International Journal of High Speed Computing **2** 311 (1990).

[46] L. L. Smarr, Ph.D. dissertation, University of Texas at Austin, (1975).

[47] J. Thornburg, Classical and Quantum Gravity, **4** 1119 (1987).

[48] K. S. Thorne, in *Second Texas Symposium on 3D Numerical Relativity* edited by R. A. Matzner 1991.

[49] J. Towns, Personal communication.

[50] R.S. Varga, *Matrix Iterative Analysis* (Prentice-Hall, Englewood Cliffs, NJ, 1962).

[51] M. Barnett, L. Shuler, R. Van de Gein, S. Gupta, D. G. Payne, J. Watts, in *Interprocessor Collective Communication Library* 1994.

[52] R. Van de Gein, Personal communication.

[53] R. M. Wald, *General Relativity* (University of Chicago Press, Chicago Ill,1984).

[54] S. Weinberg, *Gravitation, and Cosmology: Principles and Applications of the General Theory of Relativity* (Wiley and Sons, New York, 1972).

[55] J. W. York, Jr., Physica A **124**, 629 (1984).

[56] J. W. York, Jr., and T. Piran, in *The Initial Value Problem and Beyond* (University of Texas Press, Austin, TX., 1982).

[57] J. W. York, Jr. in *Sources of Gravitational Radiation* edited by L.L. Smarr (Cambridge University Press, Cambridge, England, 1979).

[58] D.M. Young, *Iterative solution of large linear systems* (Academic Press, New York, (1971).

Vita

Scott Alan Klasky was born in Philadelphia, Pennsylvania on the 15th of June, 1966, the son of Charles and Mary Klasky. The author completed his high school work at Lower Moreland High School, PA. in 1984. He entered Drexel University in 1984 and received his B.S. in physics in 1989. During this time, the author worked as a research assistant at Princeton Plasma Physics Laboratory. In September of 1989 he entered the Graduate School of The University of Texas at Austin. From 1989 to 1994 he was a member of the Center for Relativity at The University of Texas at Austin. The author is getting married to Jennifer Patton on August 20, 1994. Directly after the honeymoon, in August 1994, the author will be employed as a Postdoctoral Fellow at The University of Texas at Austin.

Permanent address: 10926 Jollyville Rd. #1124 Austin, Texas 78759.

This dissertation was typed in LaTeXby the author.