Radiative Problems in Black Hole Spacetimes

by

Robert Lee Marsa, 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 1995

Radiative Problems in Black Hole Spacetimes

Approved by
Dissertation Comittee:

To Anissa

Acknowledgements

I would like to thank my supervisor, Dr. Matthew Choptuik for his invaluable assistance throughout my graduate studies. In particular, I would like to thank him for directing me to study these problems, for help and advice in solving them, for checks of the equations, and for entertaining and useful discussions.

Special thanks also to Dr. Richard Matzner for his advice, assistance, and hospitality. Thanks to my parents for all their support, and to my wife for *everything*. And thanks to many of my graduate, undergraduate, and high-school teachers who inspired me.

This work was supported in part by the Texas Advanced Research Project TARP-085 to Richard Matzner, the Cray Research Grant to Richard Matzner, and NSF PHY9318152 (ARPA supplemented) to the Grand Challenge Alliance.

Radiative Problems in Black Hole Spacetimes

Publication No. _______________

Robert Lee Marsa, Ph.D.
The University of Texas at Austin, 1995

Supervisor: Matthew W. Choptuik

This dissertation investigates finite difference techniques which are useful for solving radiative problems in spacetimes which contain a black hole. The singularities present in such spacetimes are avoided by excising the interior of the black hole from the computational domain. The boundary of the black hole is chosen at the apparent horizon. Spatial derivatives at this boundary are tipped so that they only reference points outside the black hole. Programs using this method are used to examine the interaction of a scalar field with a Schwarzschild black hole in spherical symmetry and with a Kerr black hole in three dimensions.

The main spherically symmetric calculation looks at the scattering of ingoing packets of massless scalar field. *Quasi-normal* ringing and power-law tails are observed, along with interesting coordinate and nonlinear effects. Also examined is the stability of a static solution found by Bechmann and Lechtenfeld. This solution describes a static configuration of scalar field with potential outside a black hole.

The three dimensional calculation looks at the scattering of packets of massless scalar field from a fixed Kerr background. The phenomenon of *superradiance* is examined.

The programs used in this work were constructed using the new prototyping language RNPL. This language allows for the fairly simple construction and modification of programs to solve time-dependent partial differential equations. RNPL and its compiler are discussed near the end of this dissertation.

Contents

List of Figures

List of Tables

Chapter 1. Introduction

As the title suggests, this dissertation is concerned with solving problems involving radiation in spacetimes which contain a black hole. Due to the number and complexity of the equations of general relativity (Einstein's equations), exact solutions are hard to come by except in the simplest situations. Thus, relativists turn to computers and numerical methods to solve more realistic problems. However, computers have difficulty solving these problems as well.

Again, because of the complexity of the equations, and in particular, because of their nonlinear behavior, the Einstein equations are difficult to code in an error free and stable manner. Further, calculations involving strong fields are likely to develop black holes—and with black holes come singularities. It is important to note that these singularities are *physical* singularities near which the curvature scalar quantities experience unbounded growth. As such, they can not be removed by coordinate transformations or other methods typically used to deal with singular points in differential equations.

1.1. Black Holes, Singularities, and Horizon Boundaries

As can be expected, computers, with their finite precision, have great difficulty treating singularities. Thus, the only hope of numerically solving relativistic problems involving generic strong fields or initial black holes is to find some way to "avoid" the singularities which are either present initially or likely to develop.

A common method of avoiding singularities in gravitational collapse is to make use of the coordinate freedom allowed by general relativity to let time elapse at different rates in different parts of the spacetime. In particular, a coordinate system can be chosen so that time slows rapidly as one nears the singularity, stopping before it is reached. This prevents the region of spacetime covered by the chosen coordinate system from ever encountering the singularity. Unfortunately, this method introduces its own problems. In order to remain at rest (at a fixed coordinate location) near the singularity, an observer will have to experience an unbounded acceleration as proper time progresses. This acceleration term appears in the Einstein equations and will cause the numerical evolution to eventually halt. The space-like physical singularity has been avoided by introducing a time-like coordinate singularity caused by unbounded growth in the field variables. What is needed is a way to avoid the physical singularity without introducing any coordinate singularities.

By definition, black-hole-spacetimes contain event horizons. In fact, the *Cosmic Censorship Conjecture* states that any singularity will always be hidden within such a horizon. Although it is possible to construct certain collapse situations which result in the formation of a "naked" singularity, that is, one not hidden inside a horizon, such situations are unlikely to develop during a generic calculation. Thus we can reasonably expect any singularity that forms during a calculation will be hidden inside an event horizon. This horizon is a closed surface out of which no information can pass. This gives rise to the idea of avoiding singularities by black hole excision. We simply confine our numerical evolution to only those events outside the horizon, ignoring the region of spacetime inside the horizon altogether.

Unfortunately, this method is not without its problems either. Location of the event horizon requires knowledge of the entire spacetime—knowledge not available until the calculation is complete. But apparent horizons can be found from information about the curvature of a single space-like slice. Like an event horizon, an apparent horizon is a trapped surface—a surface through which no light can escape. Further, apparent horizons always lie within the event horizon. Thus, if we stop our numerical domain at the apparent horizon, we may evolve part of the interior of the black hole, but will still avoid any singularities that lie within. Seidel and Suen have used such a scheme with encouraging results (see [22] and [1]).

For the research in this dissertation, I use the same approach, though without the *causal differencing* used in the previous works. I look at the interactions of a massless scalar field with an existing black hole in one and three spatial dimensions. In the 1D case, I evolve the geometric variables along with the scalar field using the full Einstein-Klein-Gordon equations, while in the 3D case, I evolve the scalar field on a *fixed* Kerr background.

1.2. Computer Aided Programming

Even with a good coordinate system and an apparent horizon boundary, solving the Einstein equations is still no simple task. Finding a stable differencing scheme—especially at the boundaries—is a time consuming, iterative task. When a given implementation is unstable, it is often difficult to determine if the problem is due to a coding error or the differencing scheme. Every time the differencing scheme is changed, there is a new opportunity for introducing errors. What is needed is a computer program to assist the numerical physicist. This program should allow difference schemes to be easily changed without introducing errors. It should also assist in other ways by providing robust facilities for check-pointing and output and parameter control.

I have constructed such a system which I call RNPL—Rapid Numerical Prototyping Language. The system includes both the language and a compiler which takes RNPL programs and converts them to C or FORTRAN which can then be compiled and executed on a variety of machines. I used RNPL to write all the programs used in this research.

1.3. Notation and Conventions

Like most numerical relativists, I will be using a metric with signature (–, +, +, +) so that the 3-metric is positive definite. Latin indices $(i, j, k, \ldots)$ on tensors range over {1, 2, 3} while Greek indices $(\alpha, \beta, \gamma, \ldots)$ range over $\{0, 1, 2, 3\}$. I observe the Einstein summation convention in which repeated indices indicate a sum over the repeated index. ∇_α represents covariant differentiation with respect to the four-metric, D_α represents covariant differentiation with respect to the three-metric, and ∂_α represents ordinary partial differentiation. I adopt geometricized units in which $G = c = 1$.

As is usual when discussing numerical analysis, discretized functions use superscripts for the time index and subscripts for the spatial indices. For instance, $f^n_{i,j,k} \equiv f(n\Delta t, i\Delta x^1, j\Delta x^2, k\Delta x^3)$. Further, in discussions of finite difference equations, n will refer to the time level, while i,j, and k will refer to the spatial grid location.

Chapter 2. Equations and Methods

2.1. General $3+1$ Equations

The derivation of the $3 + 1$ form of the Einstein equations has been carried out in detail many times (see for example [24] and [18]). Here, I will simply give a brief description of the $3+1$ method and state the equations in a coordinate independent form as given in [13]. However, I will show the derivation of the Klein-Gordon equation.

In order to write the equations in $3 + 1$ form, we must break the coordinates into one "time" coordinate and three "space" coordinates. To do this, we choose a "time" function t and use it to "slice" spacetime into a set of hypersurfaces Σ_t of constant t. We also choose a "time flow" vector field t^μ satisfying $t^\mu \nabla_\mu t = 1$. If n^μ is a unit vector field normal to the Σ_t, we can decompose t^μ into parts normal and tangential to Σ_t.

$$t^\mu = \alpha n^\mu + \beta^\mu. \tag{2.1.1}$$

Equation (2.1.1) defines the *lapse* function α and the *shift* vector β^i. Given the normal n^μ, we can define the three-metric by:

$$h_{\mu\nu} = g_{\mu\nu} + n_\mu n_\nu. \tag{2.1.2}$$

The *extrinsic curvature* is defined as:

$$K_{ij} \equiv \frac{1}{2} \pounds_n h_{ij}, \tag{2.1.3}$$

where $\pounds_n$ is the Lie derivative along n^μ. Written in $3+1$ form, the line element is:

$$ds^2 = -\alpha^2 dt^2 + h_{ij}\left(dx^i + \beta^i dt\right)\left(dx^j + \beta^j dt\right). \tag{2.1.4}$$

The metric and its inverse in matrix notation are:

$$g_{\mu\nu} = \begin{pmatrix} -\alpha^2 + \beta^i\beta_i & \beta_j \\ \beta_i & h_{ij} \end{pmatrix}, \tag{2.1.5}$$

and

$$g^{\mu\nu} = \begin{pmatrix} -\dfrac{1}{\alpha^2} & \dfrac{\beta^i}{\alpha^2} \\ \dfrac{\beta^j}{\alpha^2} & h^{ij} - \dfrac{\beta^i\beta^j}{\alpha^2} \end{pmatrix}. \tag{2.1.6}$$

The covariant form of the Klein-Gordon equation is

$$\nabla^\mu\nabla_\mu\phi = \partial_\phi V(\phi), \tag{2.1.7}$$

where $V(\phi)$ is an interaction potential. This reduces to

$$\frac{1}{\sqrt{-g}}\,\partial_\mu\left(\sqrt{-g}g^{\mu\nu}\partial_\nu\phi\right) = \partial_\phi V(\phi), \tag{2.1.8}$$

where g is the determinant of the metric. If we multiply by $\sqrt{-g}$ then the right hand side is simply $\alpha\sqrt{h}\,\partial_\phi V$, where h is the determinant of the three-metric. From the left hand side, we get

$$\partial_t\left(\alpha\sqrt{h}\,g^{t\nu}\partial_\nu\phi\right) + \partial_i\left(\alpha\sqrt{h}\,g^{i\nu}\partial_\nu\phi\right) =$$

$$\partial_t\left[\alpha\sqrt{h}\left(g^{tt}\partial_t\phi + g^{ti}\partial_i\phi\right)\right] + \partial_i\left[\alpha\sqrt{h}\left(g^{it}\partial_t\phi + g^{ij}\partial_j\phi\right)\right] =$$

$$\partial_t\left[\frac{\sqrt{h}}{\alpha}\left(\beta^i\partial_i - \partial_t\right)\phi\right] + \partial_i\left[\frac{\sqrt{h}}{\alpha}\,\beta^i\left(\partial_t - \beta^j\right)\phi + \alpha\sqrt{h}\,h^{ij}\partial_j\,\phi\right] =$$

$$\partial_t\left[\frac{\sqrt{h}}{\alpha}\left(\beta^i\partial_i - \partial_t\right)\phi\right] - \partial_i\beta^i\left[\frac{\sqrt{h}}{\alpha}\left(\beta^j - \partial_t\right)\phi\right] + \partial_i\left(\alpha\sqrt{h}\,h^{ij}\partial_j\,\phi\right) = \alpha\sqrt{h}\,\partial_\phi V\,. \tag{2.1.9}$$

If we define four auxiliary variables by

$$\Pi \equiv \frac{\sqrt{h}}{\alpha}\left(\partial_t - \beta^i \partial_i\right)\phi\,, \tag{2.1.10}$$

and

$$\Phi_i \equiv \partial_i \phi\,, \tag{2.1.11}$$

then we can write (2.1.9) as an evolution equation for Π, namely

$$\partial_t \Pi = \partial_i \left(\beta^i \Pi + \alpha\sqrt{h}\left(h^{ij}\Phi_j\right)\right) - \alpha\sqrt{h}\,\partial_\phi V. \tag{2.1.12}$$

Differentiating (2.1.10) gives us evolution equations for the three Φ variables

$$\partial_t \Phi_i = \partial_i \left(\frac{\alpha}{\sqrt{h}}\,\Pi + \beta^j \Phi_j\right). \tag{2.1.13}$$

The Einstein equations in $3+1$ form are

$$R + K^2 - K_{ij}\,K^{ij} = 16\pi\rho_H\,, \tag{2.1.14}$$

$$D_i\,K^i_{\ j} - D_j\,K = 8\pi S_j\,, \tag{2.1.15}$$

$$\partial_t\,K^i_{\ j} = -D^i D_j \alpha + \alpha\left[R^i_{\ j} + K K^i_{\ j} - 8\pi S^i_{\ j} + 4\pi\delta^i_{\ j}\left(S - \rho_H\right)\right], \tag{2.1.16}$$

$$\partial_t\,h_{ij} = -2\,\alpha\,K_{ij} + D_i\beta_j + D_j\beta_i\,, \tag{2.1.17}$$

where $R^i_{\ j}$ is the Ricci tensor and R is the Ricci scalar. The source terms for the scalar field are

$$\rho_H = \frac{1}{2}\,h^{ij}\Phi_i\Phi_j + \frac{1}{2h}\,\Pi^2 + V\left(\phi\right) \tag{2.1.18}$$

$$S^i = -\frac{\Pi}{\sqrt{h}}\,h^{ij}\Phi_j \tag{2.1.19}$$

$$S^{ij} = h^{ij}\Big(-\frac{1}{2}\, h^{kl}\Phi_k\Phi_l + \frac{1}{2h}\, \Pi^2 - V\left(\phi\right) \Big) + h^{ik}h^{jl}\Phi_k\Phi_l \tag{2.1.20}$$

$$S = -\frac{1}{2}\, h^{ij}\Phi_i\Phi_j + \frac{3}{2h}\, \Pi^2 - 3V\left(\phi\right). \tag{2.1.21}$$

Substituting the source terms into equation (2.1.16), we get

$$\begin{aligned}\frac{1}{\alpha}\left(\partial_t - \beta^k\partial_k\right) K^i{}_j = {} & KK^i{}_j - \frac{1}{\alpha}\, h^{ik}\left(\partial_j\,\partial_k\alpha - \Gamma^l{}_{jk}\partial_l\alpha\right) + R^i{}_j \\ & - 8\pi\Big(h^{ik}\Phi_k\Phi_j + \delta^i{}_j V\left(\phi\right)\Big) + \frac{1}{\alpha}\left(K^i{}_k\partial_j\,\beta^k - K^k{}_j\,\partial_k\,\beta^i\right).\end{aligned} \tag{2.1.22}$$

Similarly, equation (2.1.17) becomes

$$\frac{1}{\alpha}\left(\partial_t - \beta^k\partial_k\right) h_{ij} = -2\,K_{ij} + \frac{1}{\alpha}\left(h_{ki}\partial_j\,\beta^k + h_{kj}\partial_i\,\beta^k\right). \tag{2.1.23}$$

The Hamiltonian constraint (2.1.14) becomes

$$R + K^2 - K_{ij}\,K^{ij} = 16\pi\Big(\frac{1}{2}\, h^{ij}\Phi_i\Phi_j + \frac{1}{2h}\, \Pi^2 + V\left(\phi\right)\Big) \tag{2.1.24}$$

and the momentum constraints (2.1.15) become

$$D_i\,K^i{}_j - D_j\,K = -8\pi\,\frac{\Pi}{\sqrt{h}}\,\Phi_j. \tag{2.1.25}$$

2.2. Numerical Analysis

There are many problems that can occur when trying to solve a set of partial differential equations numerically. As mentioned in Section 1.2, when an attempted solution method proves unstable, it is often difficult to determine the cause of the problem. Is it a programming error or is it the finite difference scheme?

More importantly, if the solution method appears to be stable, we need rigorous methods for deciding if it really is stable, and if so, is it in fact solving the correct set of equations.

Following [6] and [8], I will give some methods for rigorously determining the correctness and stability of a finite difference scheme.

2.2.1. Definitions

We are concerned with numerically solving a set of continuum partial differential equations using finite differencing. This means that we apply finite difference approximations of the differential operators to discretized versions of the functions. I will represent the discretized versions of functions and operators by hatted quantities, while using unhatted quantities for the continuum versions.

Given a function u and a differential operator L satisfying the equation

$$Lu = 0, \tag{2.2.1.1}$$

we define the *truncation error* by

$$\hat{\tau} \equiv \hat{L}u \tag{2.2.1.2}$$

and the solution error by

$$\hat{e} \equiv u - \hat{u}. \tag{2.2.1.3}$$

We say that the finite difference operator is pth-order accurate if $\hat{\tau} = O\left(h^p\right)$, when acting on a function discretized on a grid with spacing h. The schemes I will be using in this dissertation are all 2nd-order accurate, so for the remaining discussion I will assume $p = 2$.

If $\hat{L}$ is made up of centered difference operators, then as the grid spacing goes to zero, we get the following continuum expansion for the discretized function [21]:

$$\hat{u} = u - h^2 e_2 - h^4 e_4 - \cdots, \tag{2.2.1.4}$$

where e_i is an h-independent error function. Similarly, if $\hat{L}$ is not completely centered, we will, in general, get the following continuum expansion for the discretized function:

$$\hat{u} = u - h^2 e_2 - h^3 e_3 - h^4 e_4 - \cdots. \tag{2.2.1.5}$$

2.2.2. Methods

Assume that we have a program to generate solutions to the system $\hat{L}\hat{u} = 0$. This program appears to be stable, that is, for the time we have evolved the solution, nothing has "blown up." How can we check to make sure the method is really stable? We must check for convergence. If the solution is not convergent, then it is not stable.

To check for convergence, we must compute solutions on three grids with different resolutions. It is convenient to choose one with spacing h, one with spacing $2h$, and one with spacing $4h$. We will call these solutions $\hat{u}_h$, $\hat{u}_{2h}$, and $\hat{u}_{4h}$, respectively. From (2.2.1.4), we can see that for a centered scheme, these solutions should have the following expansions:

$$\hat{u}_h = u - h^2 e_2 - h^4 e_4 - \cdots$$

$$\hat{u}_{2h} = u - 4h^2 e_2 - 16h^4 e_4 - \cdots$$

$$\hat{u}_{4h} = u - 16h^2 e_2 - 256h^4 e_4 - \cdots.$$

Then

$$\hat{u}_{2h} - \hat{u}_{4h} = 12h^2 e_2 + 240h^4 e_4 + \cdots$$

and

$$\hat{u}_h - \hat{u}_{2h} = 3h^2 e_2 + 15h^4 e_4 + \cdots.$$

The *convergence factor* is defined as

$$C_f \equiv \frac{\hat{u}_{2h} - \hat{u}_{4h}}{\hat{u}_h - \hat{u}_{2h}}. \tag{2.2.2.1}$$

In this case, $C_f = 4 + O(h^2)$. Going through a similar series of expansions, we can see that for a 1st-order accurate scheme we get $C_f = 2 + O(h^2)$. If the difference operators are not properly centered, then we will get $C_f = 4 + O(h)$ and $C_f = 2 + O(h)$ for 2nd and 1st-order schemes.

Thus, if we've constructed a 2nd-order accurate scheme, we should expect the convergence factor to be approximately four. If we compute a convergence factor that is less than one, we know that the scheme is unstable. If we make the grid spacing small enough, the solution will "blow up." However, if we compute a convergence factor of four, then we know the scheme is stable for the data being evolved. Other data can cause the scheme to exhibit other behavior, especially if the equations are nonlinear. It is important to check any interesting or unexpected solutions for convergence to make sure they are not numerical artifacts.

The fact that a program is stable and convergent simply means that it is correctly solving the set of algebraic finite-difference equations. In order to show that it is actually approximating the desired set of differential equations, we construct another finite difference approximation to L which we will call $\tilde{L}$. We then compute $\tilde{L}\hat{u}$. For a 2nd-order, centered approximation, we should get

$$\tilde{L}\hat{u} = h^2 f_2 + h^4 f_4 + \cdots,$$

where again, f_2, f_4, etc. are h-independent error functions. We then compute the convergence factor from $\left(\tilde{L}\hat{u}_h, \tilde{L}\hat{u}_{2h}, \tilde{L}\hat{u}_{4h}\right)$. If we get four, then we can be sure that we are solving the desired set of equations. $\tilde{L}\hat{u}$ is called a *residual* and since $\hat{u}$ was found using $\hat{L}$, $\tilde{L}$ is called an *independent* residual evaluator.

Chapter 3. Spherical Symmetry I: Theory

3.1. Spherically Symmetric Einstein-Klein-Gordon Equations

The Einstein-Klein-Gordon equations (see Section 2.1) can be specialized to spherical symmetry resulting in a tremendous simplification of the system. I will adopt the usual names for spherical coordinates, namely (t, r, θ, ϕ). In this coordinate system, h_{ij} and $K^i{}_j$ are diagonal. We have

$$h_{ij} = \mathrm{diag}\left(a^2(t,r), r^2 b^2(t,r), r^2 b^2 \sin^2\theta\right) \tag{3.1.1}$$

$$K^i{}_j = \mathrm{diag}\left(K^r{}_r(t,r), K^\theta{}_\theta(t,r), K^\theta{}_\theta\right) \tag{3.1.2}$$

$$\beta^i = \left(\beta^r(t,r), 0, 0\right) \equiv (\beta, 0, 0) \tag{3.1.3}$$

$$\alpha = \alpha(t,r), \phi = \phi(t,r). \tag{3.1.4}$$

$$\Phi_i = \left(\Phi_r(t,r), 0, 0\right) \equiv (\Phi, 0, 0) \tag{3.1.5}$$

The Christoffel symbols are given by

$$\Gamma^i{}_{jk} = \frac{1}{2} h^{il}\left(\partial_k h_{lj} + \partial_j h_{lk} - \partial_l h_{jk}\right). \tag{3.1.6}$$

In spherical symmetry, the non-zero components are

$$\Gamma^r{}_{rr} = \frac{\partial_r a}{a} \qquad \Gamma^r{}_{\theta\theta} = -\frac{rb\partial_r(rb)}{a^2} \qquad \Gamma^\theta{}_{r\theta} = \frac{\partial_r(rb)}{rb}$$

$$\Gamma^{r}{}_{\phi\phi} = -\sin^2\theta \, \frac{rb\partial_r(rb)}{a^2} \qquad \Gamma^{\phi}{}_{r\phi} = \frac{\partial_r(rb)}{rb}$$
$$\Gamma^{\theta}{}_{\phi\phi} = -\sin\theta\cos\theta \qquad \Gamma^{\phi}{}_{\phi\theta} = -\cot\theta$$

The two non-zero components of the Ricci tensor are

$$R^{r}{}_{r} = -\frac{2}{arb}\,\partial_r \frac{\partial_r(rb)}{a} \tag{3.1.7}$$

$$R^{\theta}{}_{\theta} = \frac{1}{ar^2b^2}\left[a - \partial_r\left(\frac{rb}{a}\,\partial_r(rb)\right)\right]. \tag{3.1.8}$$

From now on, we will denote ∂_r by a prime and ∂_t by an over dot. Equation (2.1.23) becomes

$$\dot{a} = -a\alpha K^{r}{}_{r} + (a\beta)' \tag{3.1.9}$$

$$\dot{b} = -\alpha b K^{\theta}{}_{\theta} + \frac{\beta}{r}(r\beta)'. \tag{3.1.10}$$

For the extrinsic curvature (2.1.22) we get

$$\dot{K^{r}}_{r} = \beta K^{r}{}_{r}{}' + \alpha K^{r}{}_{r} K - \frac{1}{a}\left(\frac{\alpha'}{a}\right)' - \frac{2\alpha}{arb}\left[\frac{(rb)'}{a}\right]' - \pi\alpha\left(\frac{\Phi^2}{a^2} + V(\phi)\right) \tag{3.1.11}$$

$$\dot{K^{\theta}}_{\theta} = \beta K^{\theta}{}_{\theta}{}' + \alpha K^{\theta}{}_{\theta} K + \frac{\alpha}{(rb)^2} - \frac{1}{a(rb)^2}\left(\frac{\alpha rb}{a}(rb)'\right)' - 8\pi V(\phi). \tag{3.1.12}$$

Following [6] we change our definition for Π slightly

$$\Pi \to \frac{1}{r^2b^2\sin\theta}\,\Pi = \frac{a}{\alpha}\left(\dot{\phi} - \beta\phi'\right), \tag{3.1.13}$$

while the definition for Φ remains the same

$$\Phi \equiv \phi'. \tag{3.1.14}$$

Using these variables, (2.1.12) becomes

$$\dot{\Pi} = \frac{1}{r^2 b^2}\left[r^2 b^2 \left(\beta\Pi + \frac{\alpha}{a}\Phi\right)\right]' - 2\Pi\frac{\dot{b}}{b} - \alpha a \partial_\phi V \tag{3.1.15}$$

and (2.1.13) becomes

$$\dot{\Phi} = \left(\beta\Phi + \frac{\alpha}{a}\Pi\right)'. \tag{3.1.16}$$

The Hamiltonian constraint (2.1.24) is

$$-\frac{2}{arb}\left[\left(\frac{(rb)'}{a}\right)' + \frac{1}{rb}\left(\left(\frac{rb}{a}(rb)'\right)' - a\right)\right] + 4K^r{}_r K^\theta{}_\theta + 2K^\theta{}_\theta{}^2 =$$

$$8\pi\left(\frac{\Phi^2 + \Pi^2}{a^2} + 2V(\phi)\right) \tag{3.1.17}$$

and the momentum constraint (2.1.25) is

$$\frac{(rb)'}{rb}\left(K^\theta{}_\theta - K^r{}_r\right) - K^\theta{}_\theta{}' = -4\pi\frac{\Phi\Pi}{a}. \tag{3.1.18}$$

3.2. Minimally-Modified Ingoing Eddington-Finkelstein Coordinates

For the spherically symmetric calculations, I use a coordinate system introduced in [7] from earlier work [5]. Figure 3.1 shows how the Ingoing Eddington-Finkelstein coordinates relate to Kruskal-Szekeres coordinates. Notice that the slices all penetrate the horizon and meet the singularity.

To specify this coordinate system mathematically, we must fix the lapse and the shift. First, we introduce a "shifted" areal coordinate s defined by $s \equiv r + f(t)$ for some as yet undetermined function f. Our metric is now

$$ds^2 = \left(-\alpha^2 + a^2\beta^2\right)dt^2 + 2a^2\beta\, dt dr + a^2 dr^2 + s^2 d\Omega^2. \tag{3.2.1}$$

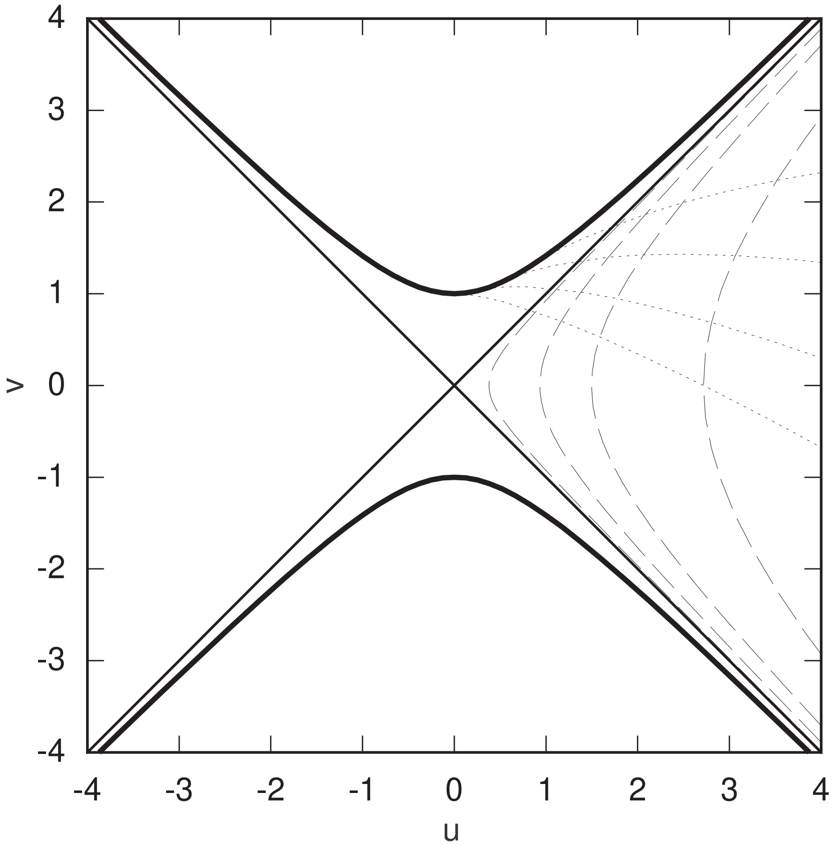

Figure 3.1. The Ingoing Eddington-Finkelstein slices in Kruskal-Szekeres coordinates. The dotted lines are constant t, while the dashed lines are constant r. The dark curves are the singularity. The diagonal lines are the horizon ($r = 2M$).

From this we see that $s = rb$ so $b = 1 + \frac{f}{r}$. Then from (3.1.10) we see that

$$\dot{\left(rb\right)} = -\alpha rbK^{\theta}{}_{\theta} + \beta\left(rb\right)' \tag{3.2.2}$$

and hence

$$\beta = \dot{f} + s\alpha K^{\theta}{}_{\theta}. \tag{3.2.3}$$

To set the lapse, we demand that the ingoing combination of tangent vectors $\vec{\partial}_t - \vec{\partial}_r$ be null. This gives a condition on the metric, namely $g_{tt} - 2g_{tr} + g_{rr} = 0$. Using (3.2.1) this implies

$$\alpha = \pm a\left(1-\beta\right). \tag{3.2.4}$$

We choose the sign so α is positive for $\left|\beta\right| \leq 1$, that is $\alpha = a\left(1-\beta\right)$. Using (3.2.3) and (3.2.4) we get

$$\beta = \frac{\dot{f} + saK^{\theta}{}_{\theta}}{1 + saK^{\theta}{}_{\theta}} \tag{3.2.5}$$

$$\alpha = \frac{a\left(-\dot{f}\right)}{1 + saK^{\theta}{}_{\theta}}. \tag{3.2.6}$$

And hence, the metric takes the form

$$ds^2 = a^2\left(2\beta - 1\right)dt^2 + 2a^2\beta\, dtdr + a^2dr^2 + s^2d\Omega^2. \tag{3.2.7}$$

Factoring the first three terms yields

$$ds^2 = a^2\Big(\left(2\beta - 1\right)dt + dr\Big)\left(dt + dr\right) + s^2d\Omega^2, \tag{3.2.8}$$

which shows that the characteristic speeds are

$$c = -1,\ 1-2\beta. \tag{3.2.9}$$

Now we are ready to write down the evolution and constraint equations in their final form. The constraints are

$$a' + \frac{1}{2s}\left(a^3 - a\right) + \frac{a^3 s}{2}K^{\theta}{}_{\theta}\left(2K^{r}{}_{r} + K^{\theta}{}_{\theta}\right) - 2\pi sa\Big(\Phi^2 + \Pi^2 + 2a^2V\left(\phi\right)\Big) = 0\,, \tag{3.2.10}$$

$$K^{\theta}{}_{\theta}{}' + \frac{K^{\theta}{}_{\theta} - K^{r}{}_{r}}{s} - \frac{4\pi\Phi\Pi}{a} = 0. \tag{3.2.11}$$

The evolution equations are

$$\dot{a} = -a^2\left(1-\beta\right)K^r{}_r + \left(a\beta\right)', \tag{3.2.12}$$

$$\dot{K^\theta}_\theta = \beta K^\theta{}_\theta{}' + a\left(1-\beta\right)\left(K^\theta{}_\theta\left(K^r{}_r + 2K^\theta{}_\theta\right) - 8\pi V\left(\phi\right)\right) + \frac{1-\beta}{s^2}\left(a - \frac{1}{a}\right) + \frac{\beta'}{as}, \tag{3.2.13}$$

$$\begin{aligned}\dot{K^r}_r = {} & \beta K^r{}_r{}' + a\left(1-\beta\right)K^r{}_r\left(K^r{}_r + 2K^\theta{}_\theta\right) + \\ & \frac{\beta-1}{a}\left[\frac{a''}{a} - \left(\frac{a'}{a}\right)^2 - \frac{2a'}{sa} + 8\pi\left(\Phi^2 + a^2 V\left(\phi\right)\right)\right] + \frac{\beta' a'}{a^2} + \frac{\beta''}{a},\end{aligned} \tag{3.2.14}$$

$$\dot{\Phi} = \left(\beta\Phi + \left(1-\beta\right)\Pi\right)', \tag{3.2.15}$$

$$\dot{\Pi} = \frac{1}{s^2}\left[s^2\left(\beta\Pi + \left(1-\beta\right)\Phi\right)\right]' - \frac{2\dot{s}}{s}\Pi - a^2\left(1-\beta\right)\partial_\phi V. \tag{3.2.16}$$

We can get an evolution equation for f from the apparent horizon equation. If s^μ is an outward-pointing, space-like unit normal to a trapped surface, then it obeys the equation [6]

$$D_i s^i - K + s^i s^j K_{ij} = 0. \tag{3.2.17}$$

In spherical symmetry, this reduces to

$$\left(rb\right)' - arbK^\theta{}_\theta = 0, \tag{3.2.18}$$

which, in MMIEF is simply

$$asK^\theta{}_\theta = 1. \tag{3.2.19}$$

To keep the horizon at fixed r, we demand that $\left.\dot{\left(asK^\theta{}_\theta\right)}\right|_{r_h} = 0$, where r_h is the initial position of the apparent horizon, $r = 2M$. This gives the following equation for $\dot{f}$:

$$\dot{f} = \left.\frac{4\pi s^2\left(\Phi + \Pi\right)^2}{a^2\left(1 - 8\pi s^2 V\left(\phi\right)\right)}\right|_{r^*}. \tag{3.2.20}$$

Since we wish to examine spacetimes which contain a black hole, it will be useful to write down the Schwarzschild solution in MMIEF coordinates. First, we note that the Ingoing Eddington-Finkelstein metric is usually written as [18]:

$$ds^2 = -\left(1-\frac{2M}{r}\right)d\tilde{V}^2 + 2d\tilde{V}dr + r^2 d\Omega^2. \tag{3.2.21}$$

Instead of using the null coordinate $\tilde{V}$, we can use a time-like coordinate defined as $t \equiv \tilde{V} - r$. With this coordinate, the metric becomes

$$ds^2 = -\left(1-\frac{2M}{r}\right)dt^2 + \frac{4M}{r}\,dtdr + \left(1+\frac{2M}{r}\right)dr^2 + r^2 d\Omega^2. \tag{3.2.22}$$

We can then set this equal to the 3 + 1 metric (3.2.7) to determine the 3 + 1 form of the Schwarzschild solution in these coordinates. The results are:

$$\alpha = \sqrt{\frac{r}{r+2M}}, \tag{3.2.23}$$

$$\beta = \frac{2M}{r+2M}, \tag{3.2.24}$$

$$a = \sqrt{\frac{r+2M}{r}}. \tag{3.2.25}$$

Using these and equations (3.1.9) and (3.1.10), we can find the extrinsic curvature components.

$$K^{\theta}{}_{\theta} = \frac{2M\left(r+2M\right)}{\left(r\left(r+2M\right)\right)^{3/2}}, \tag{3.2.26}$$

$$K^{r}{}_{r} = \frac{-2M\left(r+M\right)}{\left(r\left(r+2M\right)\right)^{3/2}}. \tag{3.2.27}$$

Finally, we note that the mass in these coordinates can be computed from the surface area as

$$M(s) = \frac{1}{2} s\left(1 - (16\pi\mathcal{A})^{-1}\mathcal{A}^{,\mu}\mathcal{A}_{,\mu}\right), \tag{3.2.28}$$

where $\mathcal{A} \equiv 4\pi s^2$. In MMIEF, this becomes

$$M(r) = \frac{1}{2} s\left(1 - \frac{1 - \left(saK^{\theta}{}_{\theta}\right)^2}{a^2}\right). \tag{3.2.29}$$

By making use of the evolution and constraint equations, we can write this mass as an integral over the mass-density. In this form we have

$$M(r) = \frac{s_h}{2} + 4\pi\int_{r_h}^{r} s^2\left(\frac{\Phi^2 + \Pi^2}{2a^2} + sK^{\theta}{}_{\theta}\frac{\Phi\Pi}{a} + V(\phi)\right)dr\,, \tag{3.2.30}$$

where r_h is the location of the apparent horizon and s_h is the value of s at r_h. Both forms of the mass will be computed numerically to provide checks on the accuracy of the program.

3.3. Regularity at the Origin

In cases where a black hole is not initially present or there is insufficient mass in the scalar field to form a black hole through collapse, the infalling matter will encounter the coordinate origin. The origin is one boundary of the computational domain, so we need boundary conditions to find the function values at this point. Since this is not a physical boundary, however, we do not have boundary values. Rather we must use regularity conditions to determine the behavior of the functions near the origin (see [2] for an extensive discussion).

3.3.1. Expansions

Since ϕ is a scalar and $a, K^{\theta}{}_{\theta}$, and $K^{r}{}_{r}$ are elements of rank-two tensors, we know they are even in r. Thus, near the origin, they have the following expansions:

$$\phi(r) = \phi_0 + \phi_2 r^2 + \phi_4 r^4 + \cdots \tag{3.3.1.1}$$

$$a(r) = a_0 + a_2 r^2 + a_4 r^4 + \cdots \tag{3.3.1.2}$$

$$K^\theta{}_\theta(r) = k_0 + k_2 r^2 + k_4 r^4 + \cdots \tag{3.3.1.3}$$

$$K^r{}_r(r) = K_0 + K_2 r^2 + K_4 r^4 + \cdots. \tag{3.3.1.4}$$

These expansions immediately give us the following conditions on the spatial derivatives:

$$\phi' = 0 \tag{3.3.1.5}$$

$$a' = 0 \tag{3.3.1.6}$$

$$K^\theta{}_\theta{}' = 0 \tag{3.3.1.7}$$

$$K^r{}_r{}' = 0\,. \tag{3.3.1.8}$$

3.3.2. Geometric Variables

Since our spacetime must be locally flat near the origin, we have $a(0) = 1$. An examination of the momentum constraint shows that $K^\theta{}_\theta = K^r{}_r$ at the origin. We can find further conditions by examining the potentially divergent terms in (3.2.13). These terms are the ones with powers of r in the denominator. When collected, they can be written as

$$\frac{(1-\beta)\left(a^2-1\right) + r\beta'}{r^2 a}\,. \tag{3.3.2.1}$$

Clearly, as $r \to 0$ both the numerator and the denominator of (3.3.2.1) go to zero. Thus we must use l'Hôspital's rule to find the correct limit. The derivative of the numerator is $2(1-\beta)\,aa' - \beta'\left(a^2-1\right) + \beta' + r\beta''$. As $r \to 0$ this goes to β'. The derivative of the denominator is $r^2 a' + 2ra$. Clearly this goes to zero as r goes to zero. Thus, we must have

$\lim_{r\to 0} \beta' = 0$.

Since we still have zero over zero we use l'Hôspital's rule again. The derivative of the numerator is $(1-\beta)\left(2aa'' + 2(a')^2\right) - \beta''(a^2-1) - 2a\beta' a' + 2\beta'' + r\beta'''$. As r goes to zero, this goes to $2(1-\beta)a'' + 2\beta''$. The derivative of the denominator is $r^2 a'' + 4ra' + 2a$. The limit of this is 2. Thus, the limit as $r \to 0$ of equation (3.3.2.1) is $(1-\beta)a'' + \beta''$.

Let us take a moment to examine the structure of β. When there is no black hole present, the shift is defined by

$$\beta = \frac{raK^{\theta}{}_{\theta}}{1 + raK^{\theta}{}_{\theta}} . \tag{3.3.2.2}$$

From this we can easily see that β is zero at the origin. Now

$$\beta' = \frac{raK^{\theta}{}_{\theta}{}' + ra'K^{\theta}{}_{\theta} + aK^{\theta}{}_{\theta}}{\left(1 + raK^{\theta}{}_{\theta}\right)^2} . \tag{3.3.2.3}$$

So,

$$0 = \lim_{r\to 0} \beta' = K^{\theta}{}_{\theta} \Rightarrow K^{\theta}{}_{\theta}(0) = 0 . \tag{3.3.2.4}$$

Now the second derivative of β is

$$\beta'' = \frac{raK^{\theta}{}_{\theta}{}'' + 2aK^{\theta}{}_{\theta}{}' + 2ra'K^{\theta}{}_{\theta}{}' + 2a'K^{\theta}{}_{\theta} + ra''K^{\theta}{}_{\theta}}{\left(1 + raK^{\theta}{}_{\theta}\right)^2} - \frac{2\left(\left(raK^{\theta}{}_{\theta}\right)'\right)^2}{\left(1 + raK^{\theta}{}_{\theta}\right)^3} . \tag{3.3.2.5}$$

As r goes to zero, this expression vanishes. Thus, $\beta''(0) = 0$.

Since $K^{\theta}{}_{\theta}$ is fixed at the origin, we must have the right hand of (3.2.13) side vanish. This will only happen if the limit of (3.3.2.1) is zero. This limit is zero only if $a'' = 0$ at the origin.

3.3.3. Scalar Field

The evolution of the scalar field is accomplished through two auxiliary variables, Φ and Π. These are defined by (3.1.14) and (3.1.13). The condition on Φ is obvious (see (3.3.1.5)), namely

$$\Phi(0) = 0. \tag{3.3.3.1}$$

The condition on Π however, is a bit more complicated. First, since the slicing condition gives $\alpha = a(1-\beta)$, we have $a\alpha^{-1} = (1-\beta)^{-1}$. Further, the shift is given by equation (3.3.2.2). Thus, we have $\Pi(0) = \dot{\phi}(0) \neq 0$. We look, then, to Π'. We have

$$\Pi' = \dot{\phi}' + \left(raK^{\theta}{}_{\theta}\right)'\left(\dot{\phi} - \phi'\right) + raK^{\theta}{}_{\theta}\left(\dot{\phi}' - \phi''\right). \tag{3.3.3.2}$$

As $r \to 0$ we see that $\Pi' \to aK^{\theta}{}_{\theta}\dot{\phi} \to 0$. Thus we have the condition

$$\Pi'(0) = 0. \tag{3.3.3.3}$$

3.3.4. Summary

We have three conditions on a

$$a(0) = 1,$$

$$a'(0) = 0,$$

$$a''(0) = 0.$$

Thus, the expansion for a is $a = 1 + a_4 r^4 + a_6 r^6 + \cdots$.

We have four conditions on the extrinsic curvature

$$K^{\theta}{}_{\theta}(0) = 0,$$

$$
\begin{aligned}
{K^\theta_\theta}'(0) &= 0, \\
K^r_r(0) &= 0, \\
{K^r_r}'(0) &= 0.
\end{aligned}
$$

Thus, the expansions for the extrinsic curvature components are $K^\theta_\theta = k_2 r^2 + k_4 r^4 + \cdots$ and $K^r_r = K_2 r^2 + K_4 r^4 + \cdots$.

And finally, we have three conditions on the scalar field variables

$$
\begin{aligned}
\Phi(0) &= 0, \\
\Pi(0) &\neq 0, \\
\Pi'(0) &= 0.
\end{aligned}
$$

Thus, the expansions are $\Phi = 2\phi_2 r + 4\phi_4 r^3 + \cdots$ and $\Pi = \dot{\phi}_0 + \dot{\phi}_2 r^2 + \cdots$.

Unfortunately, these conditions are inconsistent with the Hamiltonian constraint, equation (3.2.10). The spatial derivative of this equation gives an expression for a''. Upon taking the limit of this expression as $r \to 0$, we see that $a'' \propto \Pi^2 \neq 0$. This leads us to the conclusion that the MMIEF coordinate system will admit no non-singular curvature at the origin. The only consistent solutions near the origin are flat space or a black hole. Thus, MMIEF is only a "good" coordinate system to use when a black hole already exists in the spacetime. In a spacetime without a black hole, the equations will remain consistent as long as no energy encounters the origin. This will be the case if the scalar field is outgoing or if it collapses to form a black hole before it encounters the origin. For a collapse problem, the best thing to do would be to start with another coordinates system and change to MMIEF coordinates if a horizon forms. If no horizon forms, there is really no need for the special horizon tracking properties of MMIEF coordinates anyway.

3.4. Initial Data

In order to perform a calculation, we must have initial data for six functions $\left(\Phi, \Pi, \beta, a, K^{\theta}{}_{\theta}, K^{r}{}_{r}\right)$. These functions must satisfy three equations, (3.2.10), (3.2.11), and (3.2.5). This means that three of these functions can be arbitrarily specified.

I wish to examine the the "scattering" of compact (mostly) ingoing pulses of scalar radiation off a black hole. The word *scattering* is in quotes because scattering doesn't always occur. Long enough wavelengths ($\lambda \gg M$) will simply reflect through the origin as if the black hole were not present.

Of course it is impossible to construct a strictly ingoing pulse since the scalar field will back-scatter from its own gravitational potential. However, we can get a nearly ingoing pulse using the following method.

Let $\phi\left(r, t\right) = F\left(u \equiv r + t\right)/r$. Since the ingoing characteristic speed is one, u is an ingoing coordinate. Thus, $\dot{\phi} = \partial_u F/r$ and $\phi' = \partial_u F/r - F/r^2$. For a compact pulse we set F to a Gaussian of the form

$$F\left(u\right) = Au^2 \exp\left(-\left(u-c\right)^d/\sigma^d\right), \tag{3.4.1}$$

where d is an integer and c is the r coordinate of the center of the pulse. This results in initial data for ϕ of the form

$$\phi\left(r\right) = Ar\exp\left(-\left(r-c\right)^d/\sigma^d\right), \tag{3.4.2}$$

$$\phi'\left(r\right) = \phi\left[\frac{1}{r} - \frac{d}{\sigma^d}\left(r-c\right)^{d-1}\right], \tag{3.4.3}$$

and

$$\dot{\phi}(r) = \phi\left[\frac{2}{r} - \frac{d}{\sigma^d}\left(r-c\right)^{d-1}\right]. \tag{3.4.4}$$

These equations can be used to set Φ and Π. I solve for β, a, and $K^{\theta}{}_{\theta}$ using equations (3.2.5), (3.2.10), and (3.2.11), respectively. $K^{r}{}_{r}$ can be specified freely and the constraints can still be satisfied by adjusting the other geometric variables. However, an arbitrary $K^r{}_r$ would almost certainly not represent the desired configuration of a scalar field around a black hole.

I tried a number of methods for setting $K^r{}_r$, including finding the initial data in polar-radial coordinates and then transforming to MMIEF coordinates. However, because of the different slicings, one MMIEF slice crosses many polar-radial slices. This requires that the data be evolved in the polar-radial system for enough time so that the initial MMIEF slice is covered. I elected to use a different approach which seems to give good results. I use the Schwarzschild form (3.2.27) for $K^r{}_r$, but with varying mass. That is, I solve for $M(r)$ (3.2.30) along with the constraints and then substitute this mass profile into (3.2.27) in place of the black hole mass M.

3.5. Tails

There are a number of physical effects that we can expect to observe during the scattering of the scalar field. The first of these is the so-called power law tails (see [10] and [19]). This effect is due completely to the curvature and would occur around any mass. What happens is that outgoing radiation is back-scattered by the curvature and reaches the interior at late times. This causes the scalar field at the horizon or any fixed areal radius to fall off as a power of time, independent of the shape of the scattered pulse (see [11]).

According to this reference, given Gaussian initial data, we should expect the scalar field to go like t^{-3} at fixed areal radius and like v^{-3} at the horizon, where t is time at infinity, $v \equiv t + r^*$ is the advanced time, and

$$r^* \equiv r + 2M\ln\left(r - 2M\right) \tag{3.5.1}$$

is the "tortoise" coordinate. In MMIEF coordinates, v varies like t at the horizon, so we should expect the scalar field to fall off like t^{-3} at both the horizon and fixed areal radius.

3.6. Ringing

Another effect we should observe is *quasi-normal ringing*. This effect was observed from studying perturbations on a fixed Schwarzschild background [20]. If the perturbation field Φ is written as a sum of spherical harmonics

$$\Phi = \sum_{l} \frac{1}{r} \Psi_l(t,r)\, Y_{lm}(\theta,\phi)\,, \tag{3.6.1}$$

then the radial part will obey the Regge-Wheeler equation

$$(-\partial_t^2 + \partial_{r^*}^2)\Psi_l = V_{eff}(r)\,\Psi_l\,. \tag{3.6.2}$$

r^* is defined by (3.5.1). (3.6.2) is just a one dimensional flat-space wave equation with an effective potential

$$V_{eff}(r) = \left(1 - \frac{2M}{r}\right)\left(\frac{l(l+1)}{r^2} + \frac{2Mq}{r^3}\right), \tag{3.6.3}$$

where $q = -3$, 0, 1 for gravitational, electromagnetic, or scalar perturbations, respectively [15].

When waves impinge on the black hole, the perturbation field will oscillate at certain frequencies which depend only on the mass of the black hole. These frequencies can be found from the poles of the scattering amplitude (see [15] for a detailed treatment). The half-period for an oscillation due to a spherical scalar perturbation is $28.44M$ [12].

3.7. Mass Scaling

The final mass of the black hole should scale as a power of the initial amplitude of the scalar field. To find out what this power should be, we can use equation (3.2.30) which gives the mass as an integral of the scalar field. If we take the integral throughout space (neglecting the potential V), we get

$$M_\infty = M_h + 4\pi \int_{r_h}^{\infty} s^2 \left(\frac{\Phi^2 + \Pi^2}{2a^2} + sK^\theta{}_\theta \frac{\Phi \Pi}{a} \right) dr, \tag{3.7.1}$$

where $M_h \equiv \frac{s_h}{2}$ is the mass of the black hole. Since the mass is conserved, M_∞ is a constant. However, M_h is not constant. As the scalar field encounters the horizon, some mass will be transferred from the integral term to M_h. The mass of the black hole will increase by an amount proportional to the mass in the scalar field. For a very narrow pulse, the entire mass of the field will go into the black hole, while for a very wide pulse, almost none of it will. So, to see how the final mass of the black hole scales with the amplitude of the scalar pulse, we need only examine the integral term in equation (3.7.1).

The initial data is given by equations (3.4.2) , (3.4.3), and (3.4.4). Using these along with equations (3.1.13) and (3.1.14), we get

$$\Phi = \phi \left[\frac{1}{r} - \frac{d\left(r-c\right)^{d-1}}{\sigma^d} \right] \tag{3.7.2}$$

and

$$\Pi = \phi \left[\frac{2-\beta}{r\left(1-\beta\right)} - \frac{d\left(r-c\right)^{d-1}}{\sigma^d} \right]. \tag{3.7.3}$$

Thus, Φ and Π are both proportional to ϕ and hence to A. This means that the integrand is proportional to ϕ^2 and thus to A^2. Now this assumes that the dependence of a, $K^\theta{}_\theta$, and β on ϕ is much less than the dependence of M on ϕ which seems a reasonable assumption. However, the mass scaling will be easy to check numerically. If it turns out that $M \propto A^2$ then we know this assumption is valid.

Chapter 4. Spherical Symmetry II: Massless Scalar Field

4.1. Finite Difference Equations

I will solve equations (3.2.10) - (3.2.20) using finite difference techniques on a uniform mesh with spacings Δr and Δt.

Table 4.1 shows the operators I will use in the discretizations. Note that while the derivative operators take a lower precedence than the arithmetic operators, that is $\triangle_r f_i^{n2} = \left(f_{i+1}^{n\ 2} - f_{i-1}^{n\ 2}\right)/\Delta r$, the time averaging operator takes a higher precedence, that is $A_t f_i^{n2} = \left(A_t f_i^n\right)^2$ and $A_t\left(a_i^n b_i^n\right) = A_t a_i^n A_t b_i^n$.

I initially intended to use a free evolution scheme for this set of equations, but was unable to difference equation (3.2.14) in a stable way. Thus, I use equations (3.2.12), (3.2.13), (3.2.15), (3.2.16), and (3.2.20) to evolve $a, K^\theta{}_\theta, \Phi, \Pi,$ and f; equation (3.2.11) to find $K^r{}_r$; and equation (3.2.5) to find β.

In the interior, I use the following finite difference equations:

$$\triangle_t^d a_i^n = -A_t\left(a^2\left(1-\beta\right)\right)_i^n + \triangle_r^s\left(a\beta\right)_i^n, \tag{4.1.1}$$

$$\triangle_t^d K^\theta{}_{\theta_i}{}^n = A_t \beta_i^n \triangle_r^s K^\theta{}_{\theta_i}{}^n + A_t\left(\frac{1-\beta}{s^2}\left(a - \frac{1}{a}\right)\right)_i^n + \frac{\triangle_r^a \beta_i^n}{A_t\left(as\right)_i^n}$$

$$+ A_t\left(a\left(1-\beta\right)K^\theta{}_\theta\left(2K^\theta{}_\theta + K^r{}_r\right)\right)_i^n, \tag{4.1.2}$$

Operator	Definition	Expansion
$\triangle_r^f f_i^n$	$\left(-3f_i^n + 4f_{i+1}^n - f_{i+2}^n\right)/2\Delta r$	$\left.\frac{\partial f}{\partial r}\right\|_i^n + O\left(\Delta r^2\right)$
$\triangle_r^b f_i^n$	$\left(3f_i^n - 4f_{i-1}^n + f_{i-2}^n\right)/2\Delta r$	$\left.\frac{\partial f}{\partial r}\right\|_i^n + O\left(\Delta r^2\right)$
$\triangle_r f_i^n$	$\left(f_{i+1}^n - f_{i-1}^n\right)/2\Delta r$	$\left.\frac{\partial f}{\partial r}\right\|_i^n + O\left(\Delta r^2\right)$
$\triangle_t f_i^n$	$\left(f_i^{n+1} - f_i^n\right)/\Delta t$	$\left.\frac{\partial f}{\partial t}\right\|_i^{n+½} + O\left(\Delta t^2\right)$
$\triangle_t^d f_i^n$	$\left(f_i^{n+1} - f_i^n\right)/\Delta t +$ $\varepsilon_{dis}\left[6f_i^n + f_{i-2}^n + f_{i+2}^n - \left(f_{i-1}^n + f_{i+1}^n\right)\right]/16\Delta t$	$\left.\frac{\partial f}{\partial t}\right\|_i^{n+½} + O\left(\Delta t^2\right)$
$A_t f_i^n$	$\left(f_i^{n+1} + f_i^n\right)/2$	$f\,\|_i^{n+½} + O\left(\Delta t^2\right)$
$A_r f_i^n$	$\left(f_i^n + f_{i-1}^n\right)/2$	$f\,\|_{i-½}^n + O\left(\Delta r^2\right)$
$\triangle_r^{fa} f_i^n$	$A_t \triangle_r^f f_i^n$	$\left.\frac{\partial f}{\partial r}\right\|_i^{n+½} + O\left(\Delta r^2 + \Delta t^2\right)$
$\triangle_r^{ba} f_i^n$	$A_t \triangle_r^b f_i^n$	$\left.\frac{\partial f}{\partial r}\right\|_i^{n+½} + O\left(\Delta r^2 + \Delta t^2\right)$
$\triangle_r^a f_i^n$	$A_t \triangle_r f_i^n$	$\left.\frac{\partial f}{\partial r}\right\|_i^{n+½} + O\left(\Delta r^2 + \Delta t^2\right)$
$\triangle_r^s f_i^n$	$\left(f_i^{n+1} - f_{i-1}^{n+1} + f_{i+1}^n - f_i^n\right)/2\Delta r$	$\left.\frac{\partial f}{\partial r}\right\|_i^{n+½} + O\left(\Delta r^2 + \Delta t^2 + \Delta r \Delta t\right)$

Table 4.1. Two-Level Finite Difference Operators

$$A_t\left(\triangle_r K^\theta{}_\theta + \frac{K^\theta{}_\theta - K^r{}_r}{s} - 4\pi\frac{\Phi\Pi}{a}\right)_i^n = 0, \qquad (4.1.3)$$

$$\triangle_t^d \Phi_i^n = \triangle_r^s \Big(\beta\Phi + \big(1-\beta\big)\,\Pi \Big)_i^n, \tag{4.1.4}$$

$$\triangle_t^d \Pi_i^n = \frac{1}{A_t \big(s_i^n\big)^2} \triangle_r^s \bigg(s^2 \Big(\beta\Pi + \big(1-\beta\big)\,\Phi \Big) \bigg)_i^n - 2\,\triangle_t s_i^n A_t \Big(\frac{\Pi}{s}\Big)_i^n, \tag{4.1.5}$$

$$\triangle_t f_i^n = 4\pi A_t \left(\frac{s_i^n \left(\Phi_i^n + \Pi_i^n\right)}{a_i^n} \right)^2, \tag{4.1.6}$$

$$s_i^{n+1} = r_i + f_i^{n+1}, \tag{4.1.7}$$

$$A_t\,\beta_i^n = \frac{\triangle_t f_i^n + A_t \big(asK^\theta{}_\theta\big)_i^n}{1 + A_t \big(asK^\theta{}_\theta\big)_i^n}. \tag{4.1.8}$$

These equations are applied everywhere in the interior except at the two points next to the boundary points. At these points, I use the same equations except the dissipative time derivatives, $\triangle_t^d$ are replaced by regular time derivatives, $\triangle_t$ since the value at $i+2$ or $i-2$ is not available at these locations. It is interesting to note that all of the spatial derivatives are *angled* ($\triangle_r^s$) except for the derivative of β in equation (4.1.2) and the derivative of $K^\theta{}_\theta$ in equation (4.1.3). Switching any of these derivatives from angled to non-angled or from non-angled to angled results in an instability.

The inner boundary is fixed to the apparent horizon. Thus, there is no physical condition available for the evolution equations. Rather, due to the tipping of the light cones, the function values on the horizon can be advanced using only the points outside the black hole. Therefore, I use the same equations as I use in the interior, except the centered derivatives are replaced by forward derivatives. For example, equation (4.1.1) becomes

$$\triangle_t\, a_i^n = -A_t \Big(a^2 \big(1-\beta\big) \Big)_i^n + \triangle_r^a \big(a\beta\big)_i^n. \tag{4.1.9}$$

Since the computational grid must be finite, the outer boundary can not be extended to infinity. I adopt outgoing conditions at the outer boundary, that is, I assume that no radiation will enter the grid from large r. While this is not strictly true (there will be curvature back-scattering from the outgoing pulse), it provides a good computational solution.

For the scalar field variables Φ and Π, the outgoing conditions come from the condition on ϕ, namely $s\phi \sim F(s - ct)$, with $c = 1 - 2\beta$ being the speed of outgoing waves. This means that

$$\dot{\Phi} + (1 - 2\beta)\,\Phi' + \frac{(\dot{s} + 1 - 2\beta - 2s\beta')}{s}\,\Phi - \frac{(\dot{s} + 1 - 2\beta + 2s\beta')}{s^2}\,\phi = 0 \qquad (4.1.10)$$

and

$$(1 - \beta)\,(\Pi + \Phi) + \frac{(1 - 2\beta) + \dot{s}}{s}\,\phi = 0. \qquad (4.1.11)$$

These equations are discretized as

$$\triangle_t \Phi_i^n + (1 - 2A_t\beta_i^n)\,\triangle_r^b \Phi_i^n + \frac{\triangle_t s_i^n + 1 - 2A_t\beta_i^n - 2A_t s_i^n \triangle_r^b \beta_i^n}{A_t s_i^n} A_t \Phi_i^n$$
$$- \frac{\triangle_t s_i^n + 1 - 2A_t\beta_i^n + 2A_t s_i^n \triangle_r^b \beta_i^n}{(A_t s_i^n)^2} A_t \phi_i^n = 0 \qquad (4.1.12)$$

and

$$A_t\Big[(1 - \beta)\,(\Pi + \Phi)\Big]_i^n + \frac{1 - 2A_t\beta_i^n + \triangle_t s_i^n}{A_t s_i^n} A_t \phi_i^n = 0. \qquad (4.1.13)$$

We can get approximate conditions on a and $K^\theta{}_\theta$ from their Schwarzschild forms (3.2.25) and (3.2.26) and the integral expression for the mass (3.2.30). Outside of any matter (very weak scalar field), a and $K^\theta{}_\theta$ should take on their Schwarzschild forms. For large s we can take asymptotic expansions of these to get

$$a \sim 1 + \frac{M}{s} + O(s^{-2}) \qquad (4.1.14)$$

and

$$K^{\theta}{}_{\theta} \sim \frac{2M}{s^2} + O\left(s^{-3}\right) \tag{4.1.15}$$

Thus, at large r we have basically

$$s\left(a-1\right) \sim M \tag{4.1.16}$$

and

$$s^2 K^{\theta}{}_{\theta} \sim M. \tag{4.1.17}$$

Now how does M behave in the large s weak-field limit? Since $a \to 1$ and $K^{\theta}{}_{\theta} \to 0$ we have

$$M \sim 4\pi \int s^2 \left(\Phi^2 + \Pi^2\right) ds\,. \tag{4.1.18}$$

From the condition on ϕ we can see that $\Phi \sim G\left(u\right)/s$ and $\Pi \sim G\left(u\right)/s$, where $u \equiv s - c\,t$. Thus

$$M \sim 8\pi \int G^2\left(u\right) du \sim H\left(U\right), \tag{4.1.19}$$

that is, M is "outgoing" at large s. Therefore we get the following conditions for a and $K^{\theta}{}_{\theta}$

$$s\left(a-1\right) \sim H\left(s-ct\right) \tag{4.1.20}$$

and

$$s^2 K^{\theta}{}_{\theta} \sim H\left(s-ct\right). \tag{4.1.21}$$

These are discretized as

$$\triangle_t \Big(s\left(a-1\right)\Big)^n_i + \left(1 - 2A_t\,\beta^n_i\right) \triangle^b_r \Big(s\left(a-1\right)\Big)^n_i = 0 \tag{4.1.22}$$

and

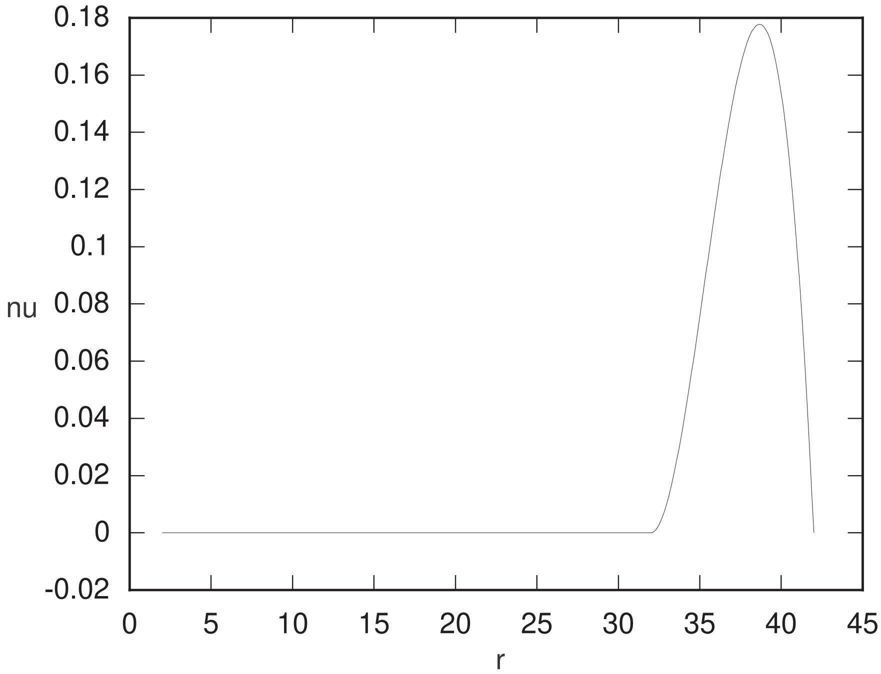

Figure 4.1. Sponge filter coefficient function for $A = 1.0$ and $n = 2$.

$$\triangle_t \left(s^2 K^{\theta}{}_{\theta}\right)_i^n + \left(1 - 2A_t \beta_i^n\right) \triangle_r^b \left(s^2 K^{\theta}{}_{\theta}\right)_i^n = 0. \tag{4.1.23}$$

The outgoing boundary condition reduces the amplitude of reflections off the boundary, but unless the boundary is placed at very large r, these reflections can still interfere with the results of a calculation. To minimize the reflections, I use a *sponge filter* as detailed in [6]. This means that in the interior of the grid, I apply the usual evolution equation along with a coefficient times the outgoing condition. For instance, in the case of Φ, the equation is

$$\begin{aligned} \dot{\Phi} = {} & \left(\beta\Phi + \left(1-\beta\right)\Pi\right)' \\ & - \nu\left[\dot{\Phi} + \left(1-2\beta\right)\Phi' + \frac{\dot{s} + 1 - 2\beta - 2\,s\beta'}{s}\,\Phi - \frac{\dot{s} + 1 - 2\beta + 2\,s\beta'}{s^2}\,\phi\right], \end{aligned} \tag{4.1.24}$$

where $\nu\left(r\right)$ is the coefficient function given by

$$\nu(r) = \begin{cases} 0 & r_{\min} \leq r < r_s \\ A\left(r - r_s\right)^n \left(r_{\max} - r\right) \left(r_{\max} - r_s\right)^{-n-2} \left(n+1\right) \left(n+2\right) & r_s \leq r < r_{\max} \end{cases}. \tag{4.1.25}$$

Here, A and n are parameters. Figure 4.1 shows ν for $A = 1.0$ and $n = 2$, the values used in this thesis.

4.2. Initial Data

Section 3.4 gives the equations used to compute the initial data. These equations are solved using an iterative procedure. First, the scalar field is set to an "ingoing" pulse (see Section 3.4) and the geometric variables are set to their Schwarzschild values (see Section 3.2). Then a and $K^{\theta}{}_{\theta}$ are integrated from equations (3.2.10) and (3.2.11). Using these, the new forms of $M(r)$ and β are computed. Finally, $K^{r}{}_{r}$ is computed. The program then computes a and $K^{\theta}{}_{\theta}$ again and so on until each of the geometric functions converges to a final value. In practice, this takes about twenty iterations.

The resulting initial data is shown in Figures 4.2 and 4.3. It seems to behave as a strictly ingoing pulse, however, it may contain an outgoing piece which only shows up at large amplitudes (see Section 4.6).

4.3. Tails

Figure 4.4 shows ϕ at constant r for runs with $r_{\max} = 42,\ 82,\ 162$. It is clear that the position of the outer boundary has a large effect on the fall-off of the scalar field, even with the sponge filter. There is enough reflection to cause the field to fall off much more slowly than it should. In order to accurately measure the tails, it would be necessary to either use an adaptive scheme so that the boundary can be moved out to several thousand M, or to match the interior evolution to a characteristic scheme which would evolve the region of spacetime from the boundary to spatial infinity.

However, with the outer boundary at $r_{\max} = 162$, it will take at least $300M$ for reflections from the scattered pulse to travel in from the outer boundary and interfere with measurements

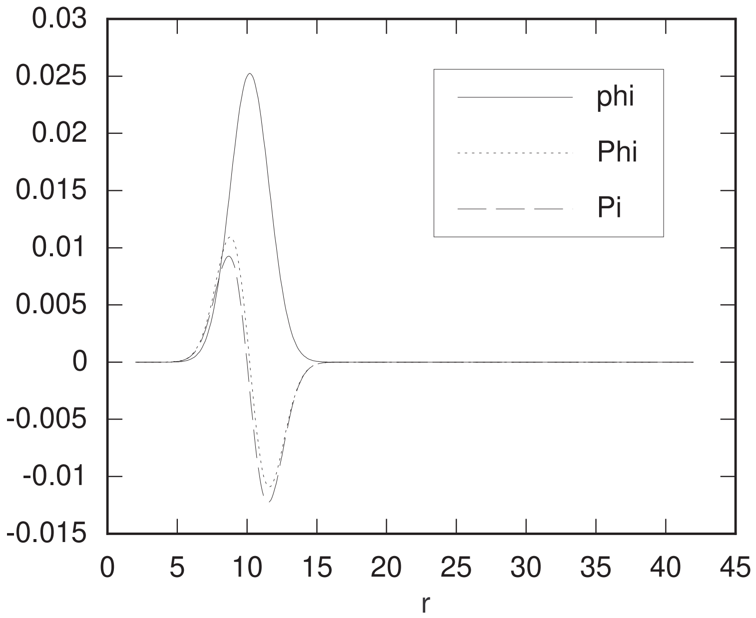

Figure 4.2. Initial data for the scalar field with $A = 3.0 \times 10^{-3}$, $c = 10$, and $\sigma = 2$.

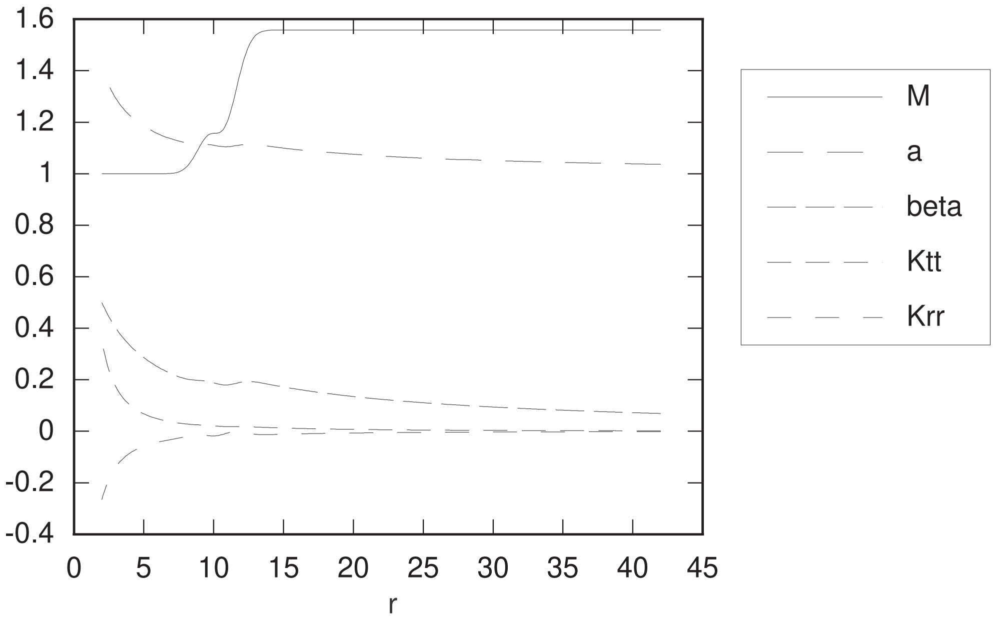

Figure 4.3. Initial data for the geometric variables and the mass profile with $A = 3.0 \times 10^{-3}$, $c = 10$, and $\sigma = 2$.

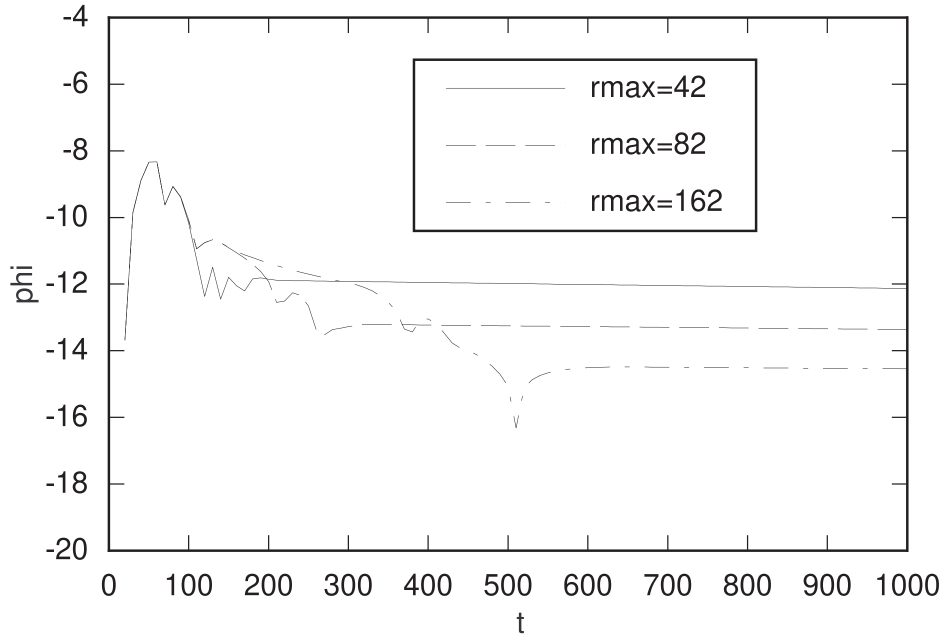

Figure 4.4. $\log\left|\phi\right|$ at $r = 30$ verses t for various spatial domains.

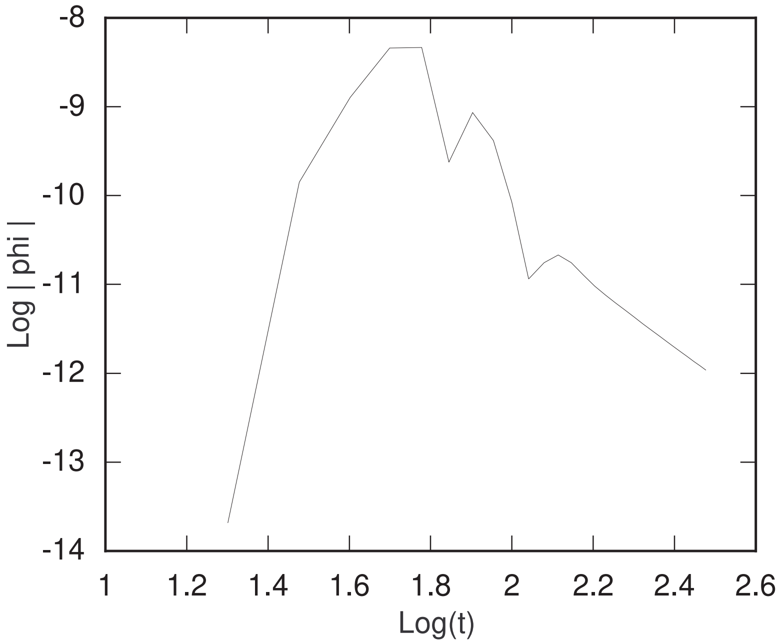

Figure 4.5. $\log\left|\phi\right|$ at $r = 30$ verses $\log t$.

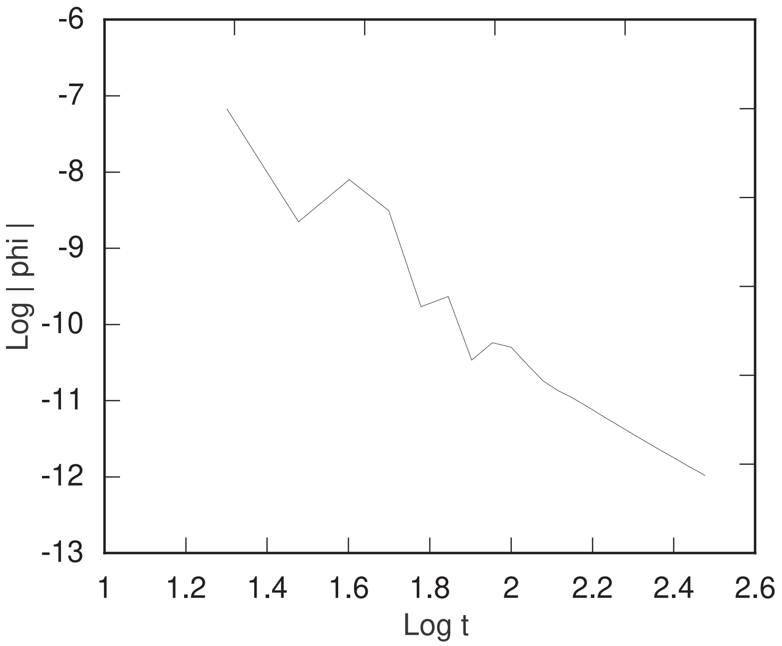

Figure 4.6. $\log|\phi|$ at the horizon verses $\log t$.

at $r = 30$. This should give enough time to measure the rate of fall off of the scalar field. Figure 4.5 shows the accurate part of the evolution. A fit to this curve between $200\,M$ and $300\,M$ shows ϕ falling off as $t^{-3.38}$. A run at twice the resolution yields the same exponent.

The evolution of ϕ at the horizon is shown in Figure 4.6. A linear fit to this curve between $200\,M$ and $300\,M$ shows ϕ falling off as $t^{-3.06}$ for runs at both resolutions.

4.4. Ringing

Figure 4.7 shows the waveforms generated by packets of various widths for medium field data ($.017M \leq M_\phi \leq .052M$). This graph shows the initial pulse of reflected scalar field. Figure 4.8 shows $\log|\phi|$ at the horizon for the same data. In this graph the subsequent oscillations are apparent. It is also apparent that the frequency is independent of the pulse width. The period for one oscillation is approximately $53\,M$ giving a half-period of $26.5M$ which is close to the predicted value of $28.44M$. A Higher resolution run with the same data yields a half-period of $26.25M$.

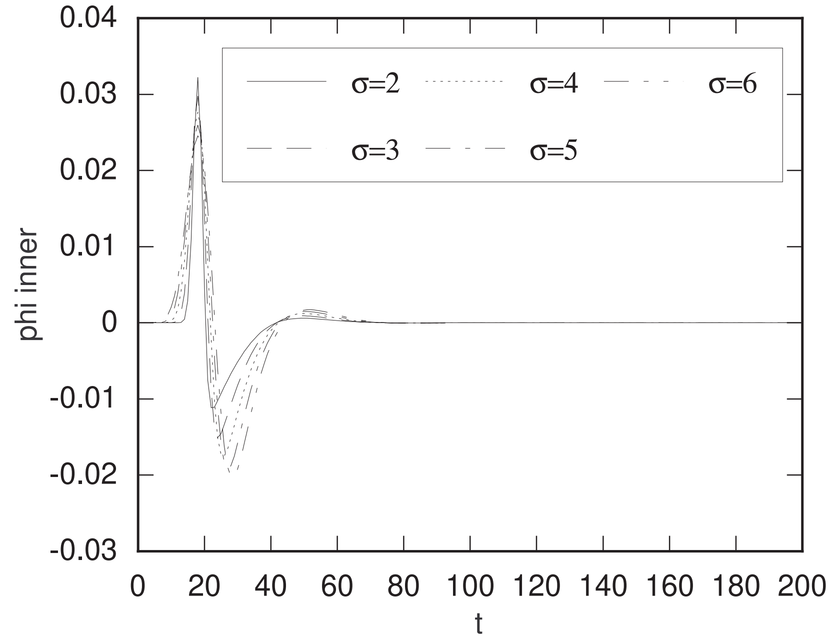

Figure 4.7. ϕ at the horizon verses time for various pulse widths ($A = 2.0 \times 10^{-4}$).

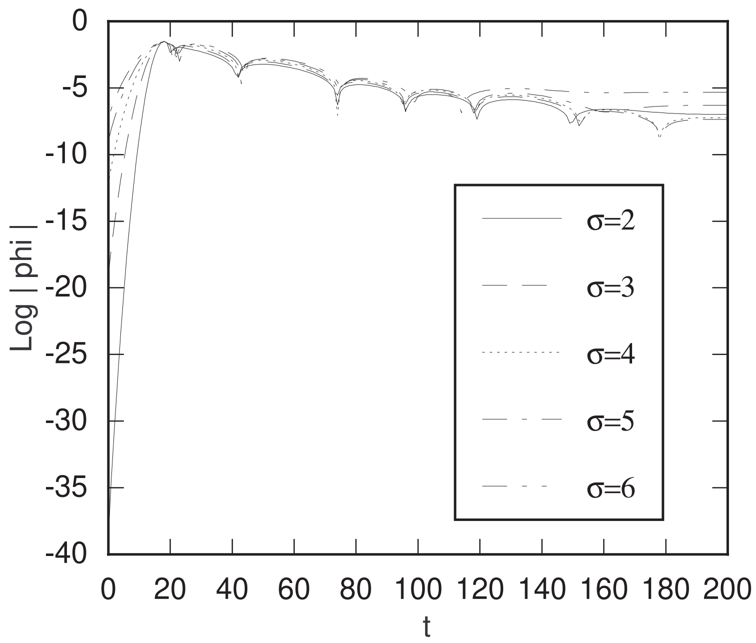

Figure 4.8. $\log|\phi|$ at the horizon verses time for various pulse widths ($A = 2.0 \times 10^{-4}$).

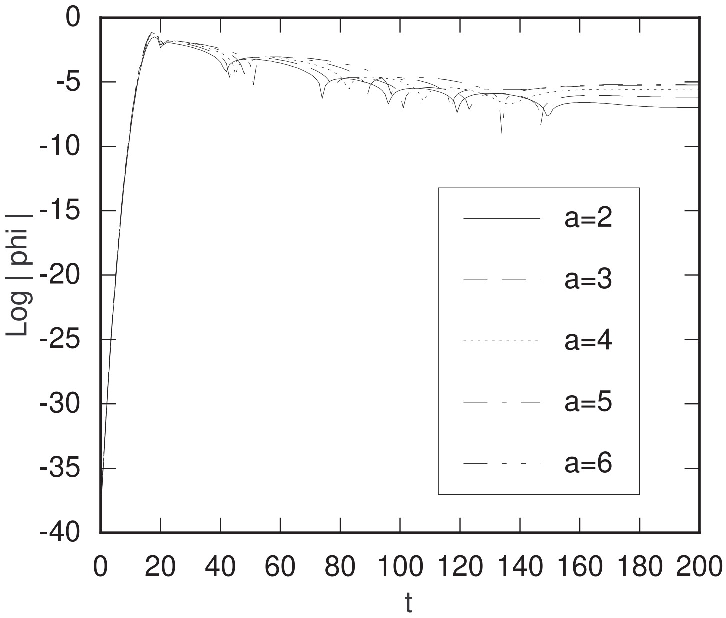

Figure 4.9. $\log\left|\phi\right|$ at the horizon verses time for various pulse amplitudes ($\sigma = 2.0$, amplitudes are $\times 10^{-4}$).

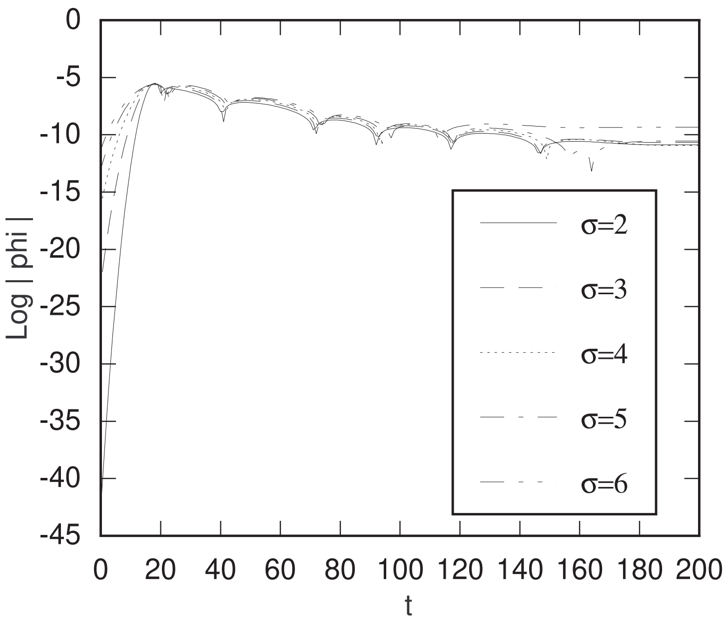

Figure 4.10. $\log\left|\phi\right|$ at the horizon verses time for various pulse widths ($A = 2.0 \times 10^{-8}$).

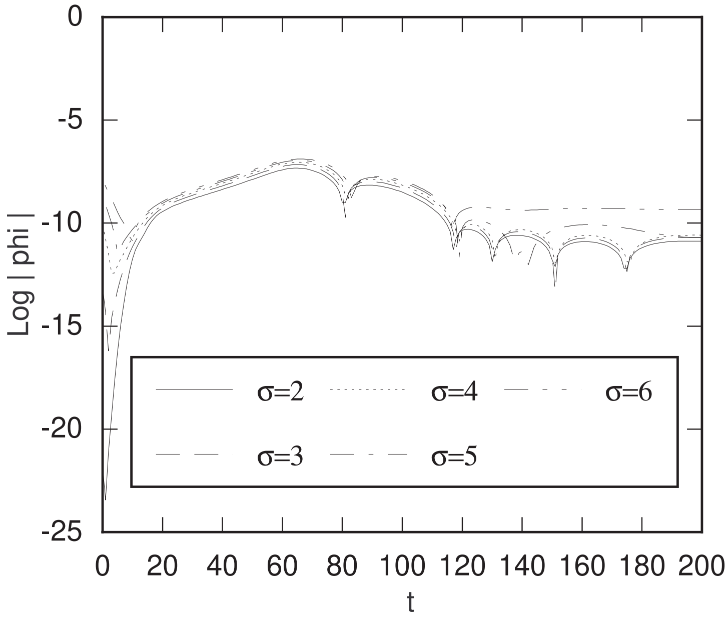

Figure 4.11. $\log|\phi|$ at $r = 30$ verses time for various pulse widths ($A = 2.0 \times 10^{-8}$).

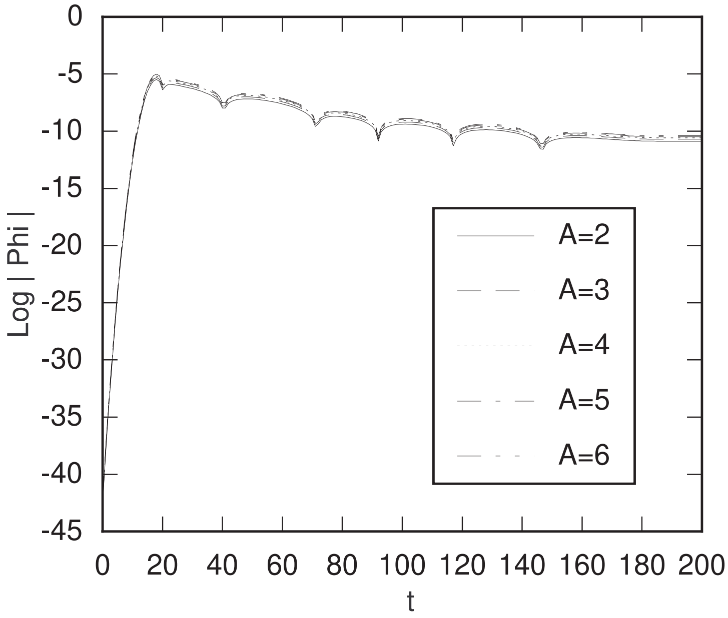

Figure 4.12. $\log|\phi|$ at the horizon verses time for various pulse amplitudes ($\sigma = 2.0$, amplitudes are $\times 10^{-8}$).

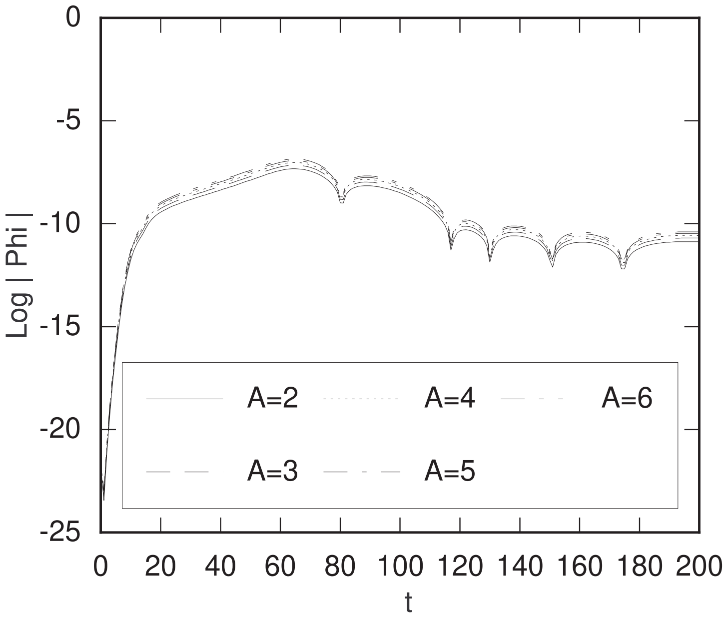

Figure 4.13. $\log|\phi|$ at $r = 30$ verses time for various pulse amplitudes ($\sigma = 2.0$, amplitudes are $\times 10^{-8}$).

Figure 4.9 shows the waveforms generated by pulses of varying amplitude for strong field data ($.051M \leq M_\phi \leq .47M$). In this case, the frequency of oscillations decreases with increasing amplitude. However, the mass of the black hole changes from 1.0 to 1.46 during the evolution of the strongest data, so the period is expected to increase.

The weak-field data shown in Figures 4.10–4.13 gives an oscillation period of approximately $53M$. This period is independent of the initial pulse amplitude as expected. For large widths, the late-time waveform differs from that of a small width. As the width becomes larger, less and less of the initial pulse is absorbed by the black hole. This means there is more scalar field available to be reflected from the outer boundary and cause differences in the late-time evolution at fixed radius.

4.5. Mass Scaling

The infalling scalar field can exhibit two main behaviors depending on the amplitude and width of the pulse. These are: *scattering* from the existing black hole and *collapse* to form

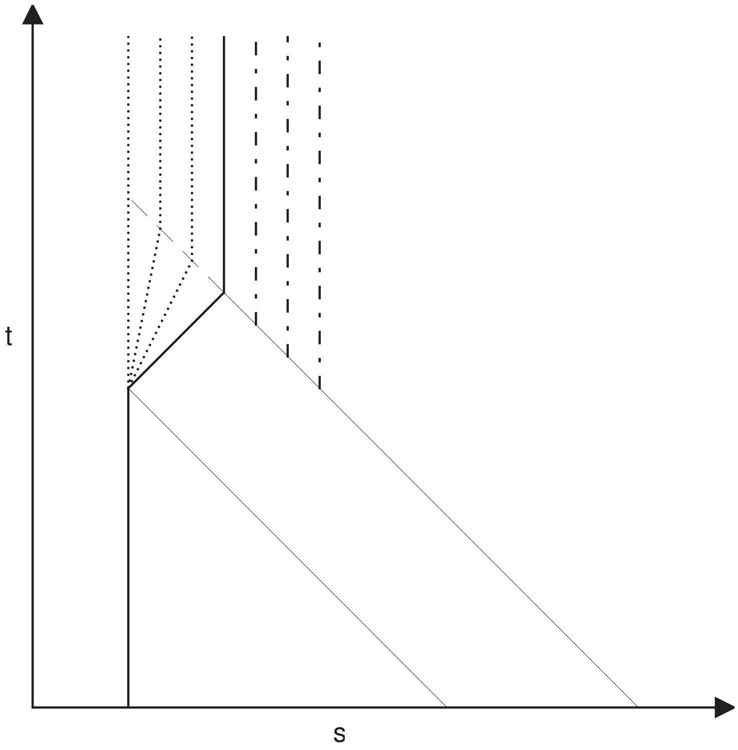

Figure 4.14. Schematic motion of the horizon for various amplitudes of the scalar field.

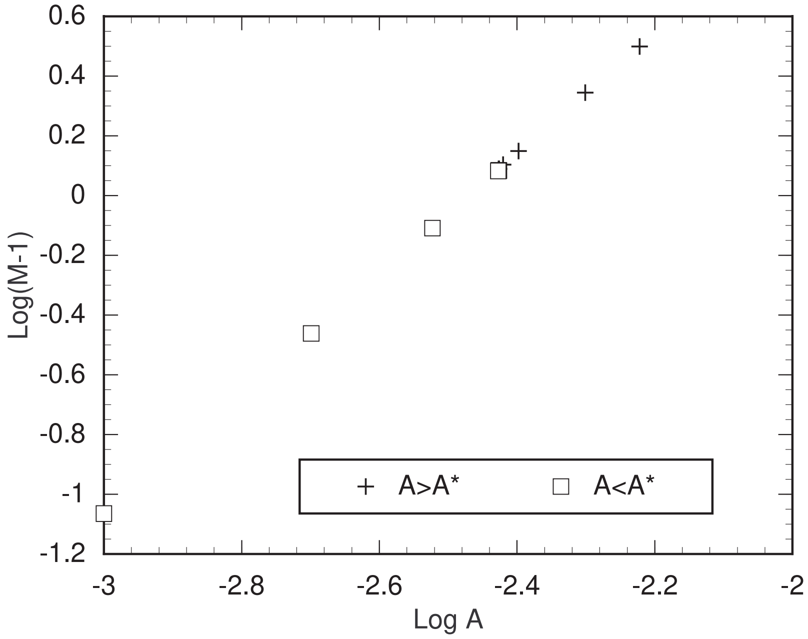

Figure 4.15. Final black hole mass verses amplitude of the scalar field pulse ($d = 2$).

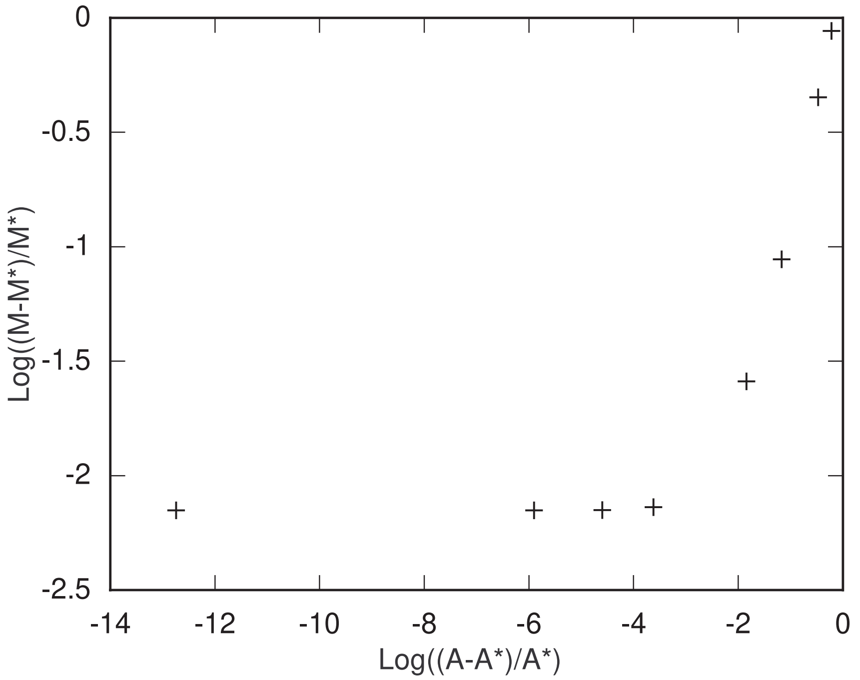

Figure 4.16. Final black hole mass verses amplitude of the scalar field pulse ($d = 2$) for super-critical amplitudes.

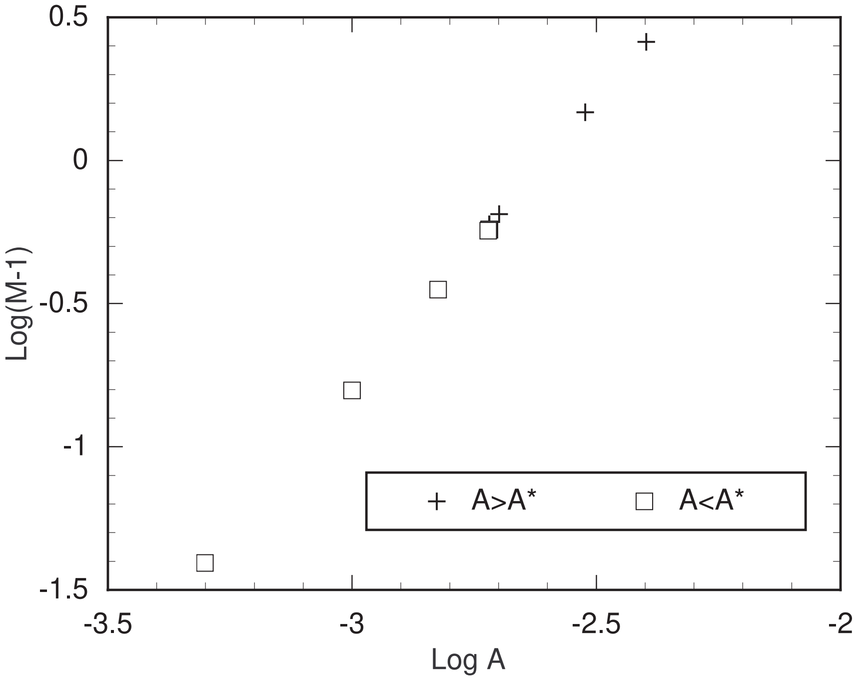

Figure 4.17. Final black hole mass verses amplitude of the scalar field pulse ($d = 4$).

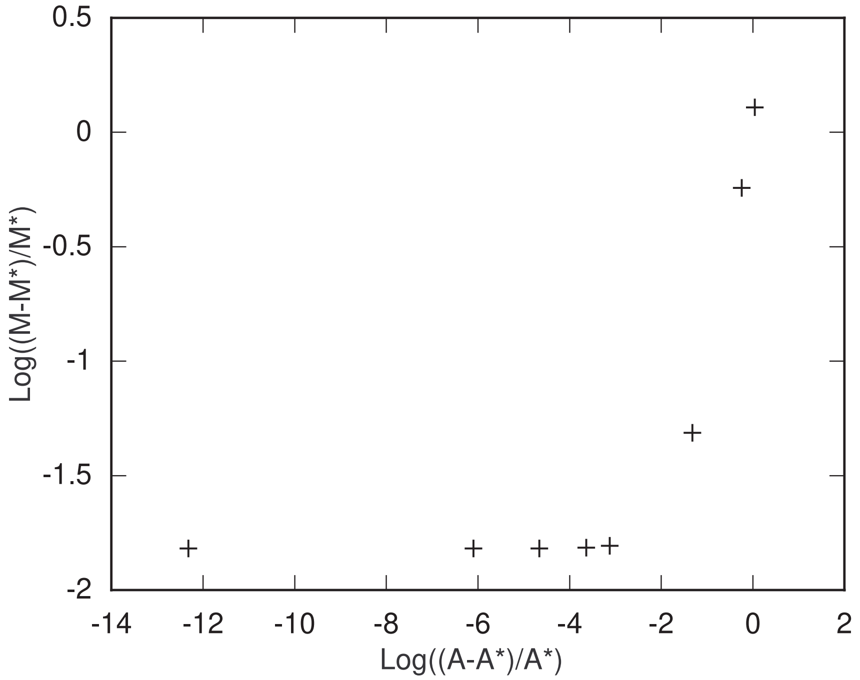

Figure 4.18. Final black hole mass verses amplitude of the scalar field pulse ($d = 4$) for super-critical amplitudes.

a new horizon outside the existing horizon. These two behaviors are separated by a *critical* value of either amplitude or width. For initial data with $\sigma = 2$, $c = 10$, and $d = 2$, the critical amplitude is $A \approx .0037$, while for initial data with $d = 4$, the critical amplitude is $A \approx .0019$.

Figure 4.14 shows a spacetime diagram of the motion of the horizon for various amplitudes of initial data. Notice that this diagram uses the areal coordinate s and not the radial coordinate r. The solid dark vertical line which jogs right and then continues vertically represents the critical path of the horizon. The dotted lines are sub-critical paths and the vertical dashed lines are super-critical paths. The two thin, diagonal lines represent the bounds of the ingoing pulse of scalar field. A super-critical pulse moves inward until it crosses its gravitational radius. Once this happens, the apparent horizon jumps from its initial position to this new position where it remains. A sub-critical pulse moves inward until it encounters the horizon. If the field is very weak, the horizon is unaffected. For stronger fields, the horizon moves out until the pulse is entirely inside. For a critical pulse, the horizon moves out at the speed of light. Note however, that unless the energy density is a square wave, the horizon will not move along the straight lines as shown in the diagram, but will move along a curve with gradually increasing and then decreasing slope.

The final mass of the black hole should scale with the amplitude of the initial data as shown in Section 3.7. If the picture in Figure 4.14 is correct, both super-critical and sub-critical data should exhibit the same mass scaling. Figure 4.15 shows $\log(M-1)$ verses A for initial data with $d=2, \sigma=2$, and $c=10$. The squares are for data with amplitude less than the critical value, while those with crosses are for data with amplitude greater than the critical value. This graph is fit by the line

$$\log(M-1) = 2.01\log A + 4.96, \tag{4.5.1}$$

indicating that the mass grows with the square of the amplitude as expected. The graph also shows there is no difference in behavior for sub- and super-critical data. That is, the final mass of the black hole exhibits the same dependence on the amplitude when the hole grows by accretion or when it forms by collapse.

Figure 4.16 shows only super-critical data. The mass values are asymptoting to $M \approx 2.22$ indicating a mass gap between the smallest super-critical black hole and the largest sub-critical black hole. In fact, the super-critical mass is $M_+ = 2.2281725$ while the sub-critical mass is $M_- = 2.2125908$. This effect is purely numerical. The method for solving the difference equations demands that the horizon be located on a grid point. The radial distance between grid points, Δr is also the areal distance Δs. The mass of the black hole is $s_h/2$ (see equation (3.2.30)). Now the location of the horizon should be accurate to within $\Delta r/2$. Thus, we would expect the mass gap to be approximately $\Delta r/4$. The data plotted in the figures was computed on a grid with $\Delta r = .1$. Thus, the mass gap should be about $\Delta M \approx .025$. The actual mass gap is about .0155. For a grid with $\Delta r = .05$ we get $M_+ = 2.316354$ and $M_- = 2.309658$ for a difference of $\Delta M \approx .0067$, while a grid with $\Delta r = .025$ gives $M_+ = 2.3660016, M_- = 2.3635547$, and $\Delta M \approx .0024$.

Figures 4.17 and 4.18 show similar behavior for data with $d=4$. Figure 4.17 is fit by the line

$$\log(M-1) = 1.99\log A + 5.19, \tag{4.5.2}$$

indicating again that the mass grows with the square of the amplitude.

Figure 4.18 again shows a mass gap. Of course this too is numerical and shrinks with the grid spacing. The grid with $\Delta r = .1$ gives $\Delta M \approx .024$, the grid with $\Delta r = .05$ gives $\Delta M \approx .006$, and the grid with $\Delta r = .025$ gives $\Delta M \approx .002$.

4.6. Coordinate Effects

The evolutions exhibit some interesting effects which are due to the use of MMIEF coordinates. The shift is given by (3.2.5). At the horizon, (3.2.19) holds, so we have

$$\beta = \frac{\dot{f}}{2} + \frac{1}{2}. \tag{4.6.1}$$

From (3.2.9) we can see that when no matter is crossing the horizon, $\beta = .5$ so the outgoing characteristic speed is zero. However, if $\dot{f} = 1$, then $\beta = 1$ and the outgoing characteristic speed is -1. In this case, the light cone is degenerate. In fact, from (3.2.5) we can see that if $\dot{f} = 1$, then $\beta = 1$ *everywhere.* Does $\dot{f}$ ever equal one? The most likely place for this to happen is the critical solution because that is when the "maximum" amount of energy is crossing the horizon for a given pulse shape. The values of β at the horizon and at the outer boundary ($r = 42\,M$) are plotted in Figure 4.19. This is for the critical solution with $\sigma = 2$, $d = 2$, and $c = 10$. β gets up around .95, but never reaches 1. The critical solution for pulses with $d = 4$ gives a slightly higher maximum β, but still less than one. I think it is likely that a narrow enough pulse could cause $\dot{f}$ to reach one for an instant, but this has not been verified.

Whenever $\beta > .5$, the outgoing characteristic speed is negative. Thus, outgoing pulses will appear to move inward. Figure 4.20 shows an evolution of $\frac{dM}{dr}$ for the critical solution referred to above. The frames are spaced $1\,M$ apart in time. The vertical scale changes at $t = 5\,M$ so the outgoing pulse can be observed. The vertical lines passing through the frames are to provide a common horizontal reference so the retrograde motion of the outgoing pulse

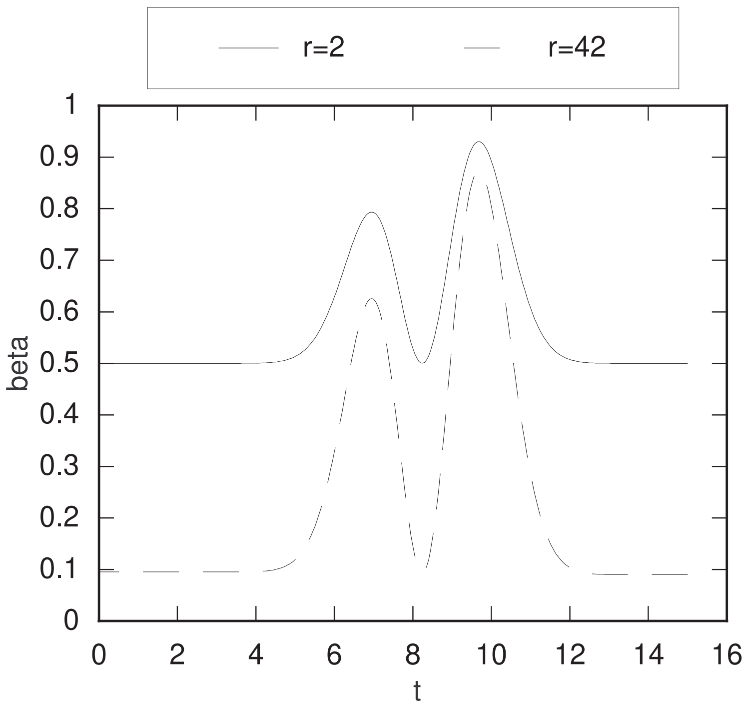

Figure 4.19. β at the horizon and the outer boundary for the critical solution with $\sigma = 2$, $d = 2$, and $c = 10$.

can be seen. There are two periods of backwards motion; one at about $7\,M$ and the other at about $10\,M$. These are the times when each of the "bumps" crosses the horizon. The retrograde motion is easier to see in Figure 4.21. This figure shows contours on an r verses t plot for the same evolution. Figure 4.22 shows a fairly weak field evolution of $\frac{dM}{dr}$ for comparison. There is no retrograde motion in this case.

4.7. Nonlinear Effects

There is a sharp "bump" at the front of the outgoing pulse in Figure 4.20. This feature is absent from the weak-field evolution of Figure 4.22 and is certainly amplitude dependent. Figure 4.23 shows a series of initial pulse shapes for data with various amplitudes and $\sigma = 2$, $d = 2$, and $c = 10$. Figure 4.24 shows the corresponding pulse shapes after scattering. The

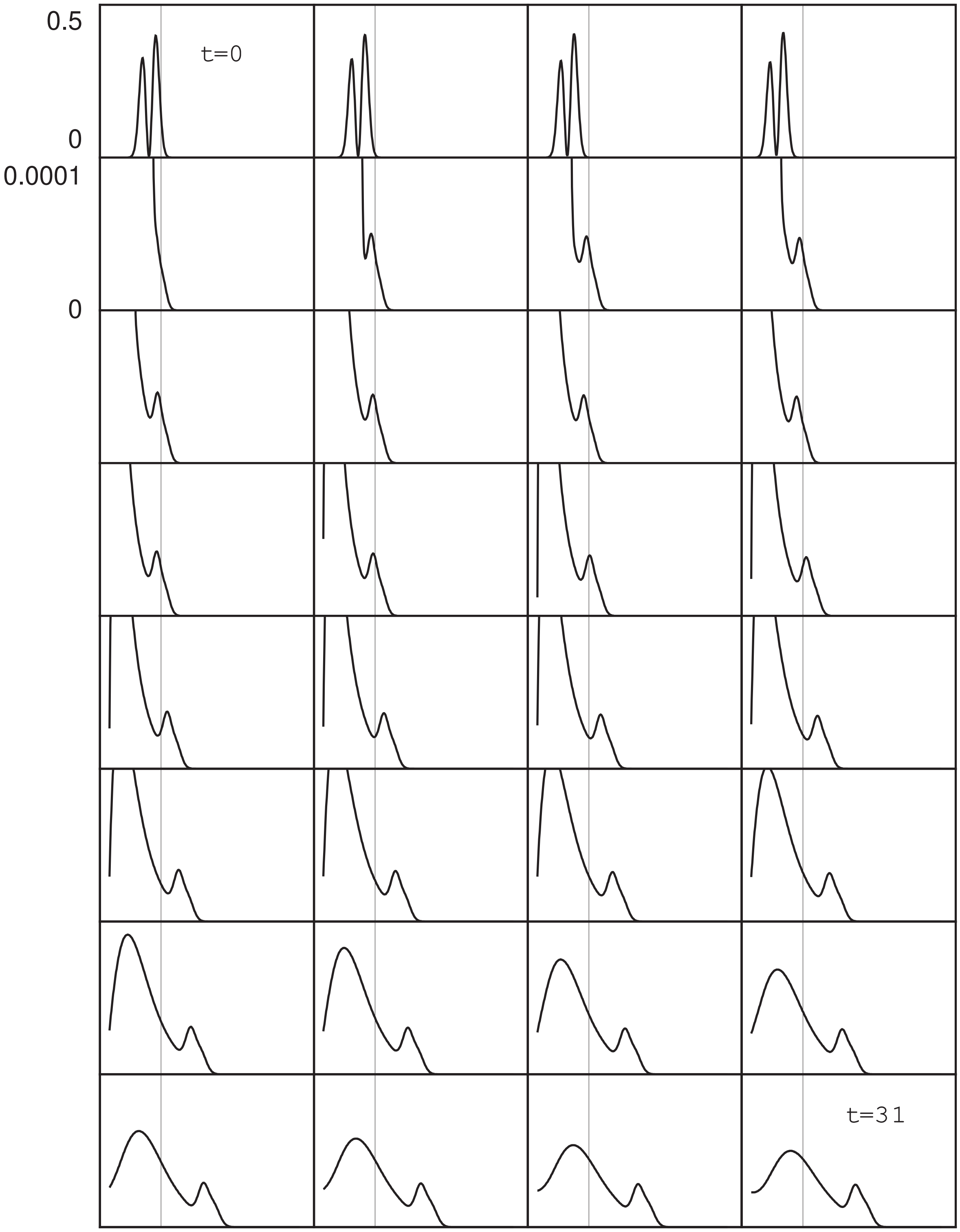

Figure 4.20. Evolution of $\frac{dM}{dr}$ for the critical solution with $\sigma = 2$, $d = 2$, and $c = 10$.

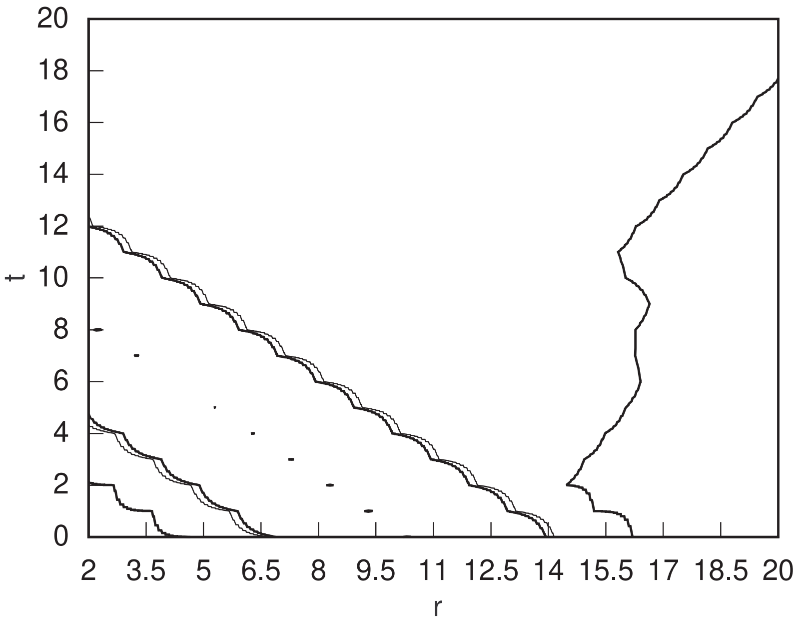

Figure 4.21. Contour plot of $\frac{dM}{dr}$ for the critical solution with $\sigma = 2$, $d = 2$, and $c = 10$.

inset in this figure lists the fraction of mass scattered times 1000. For example, the critical solution ($A = A^*$) has .243% of its mass scattered.

Although this bump occurs at the front of the scattered pulse, it is not caused by interactions with the black hole. To see this more clearly, we can start the pulse farther out and see what happens. Figures 4.25 and 4.26 show pulse shapes at $t = 0$ and $t = 8$ for data with $c = 20$. The outgoing bump develops for large amplitude data without any help from the black hole. A similar bump develops for initial data with $d = 4$ as well. These bumps always have the same characteristic shape, though their widths may vary.

It is clear from Figure 4.24 that the amplitude of this outgoing feature does not depend linearly on the initial amplitude of the pulse. For instance, from Figure 4.23, the height of the highest peak is about 1.5 times the height of the next highest peak, while their widths are the same. However, the amplitude of the outgoing feature in the first case is about 2.5 times the amplitude of the outgoing feature in the second case, while their widths are the same. Figures 4.25 and 4.26 show similar nonlinear behavior for the pulses centered at $r = 20$. It is difficult

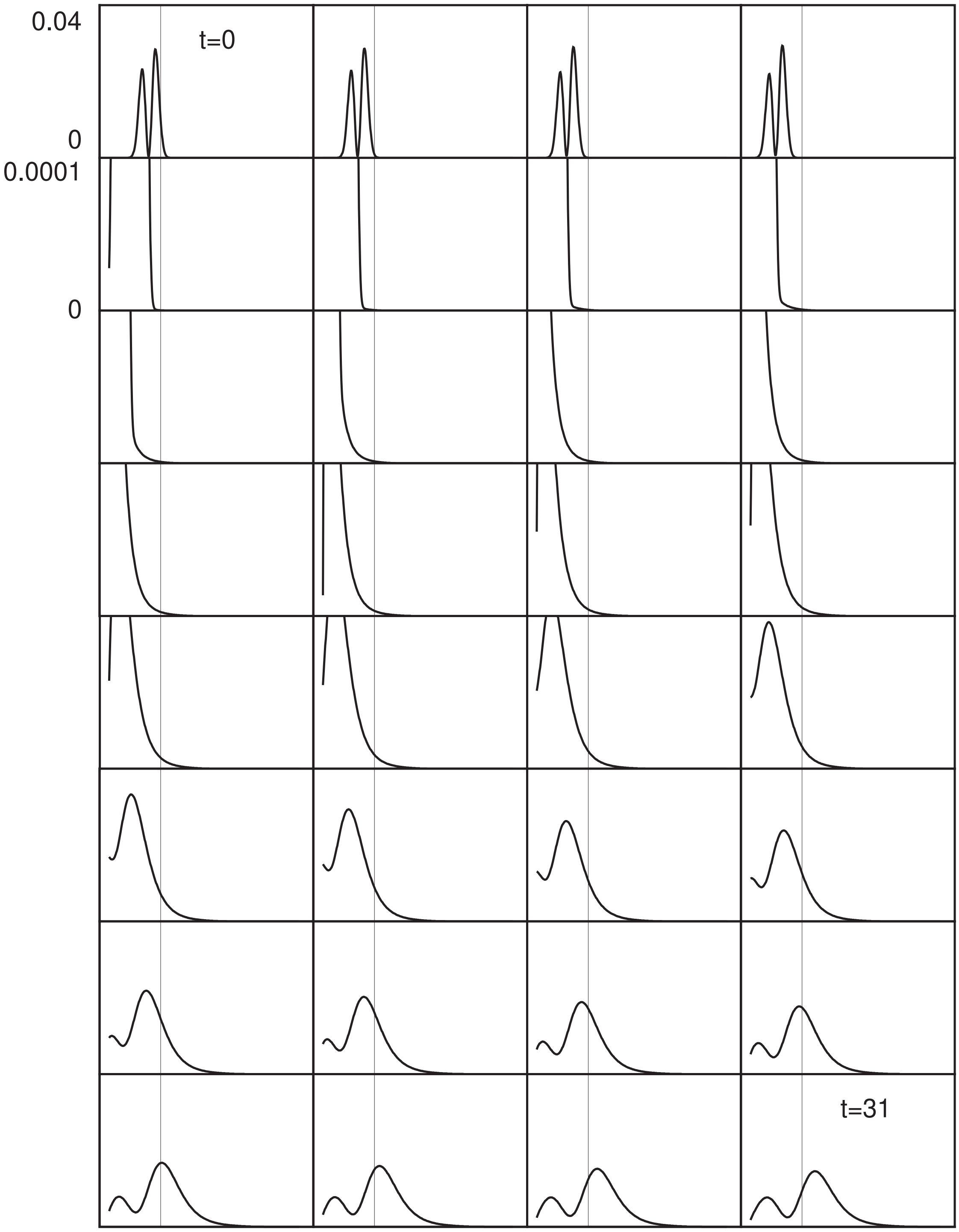

Figure 4.22. Evolution of $\frac{dM}{dr}$ for the weak-field solution with $A = .001$, $\sigma = 2$, $d = 2$, and $c = 10$.

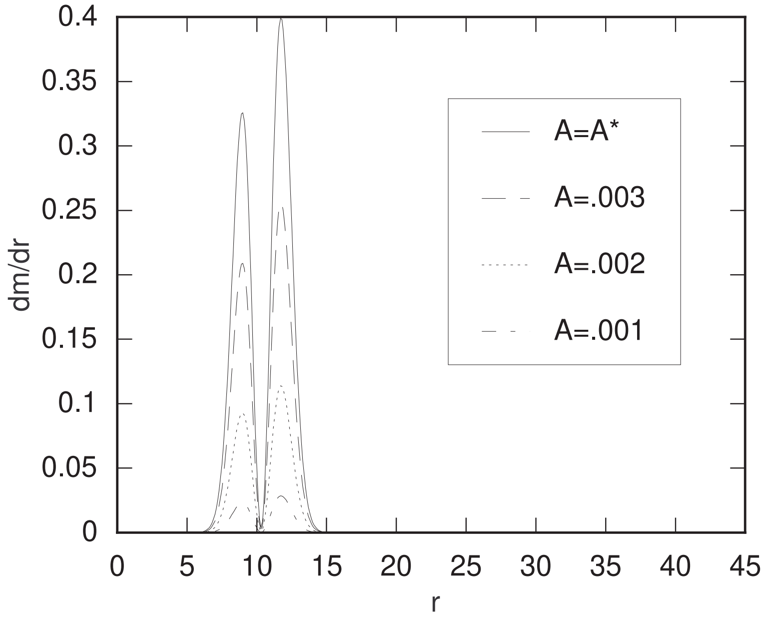

Figure 4.23. $\frac{dM}{dr}$ at $t = 0$ for $\sigma = 2$, $d = 2$, $c = 10$, and various amplitudes.

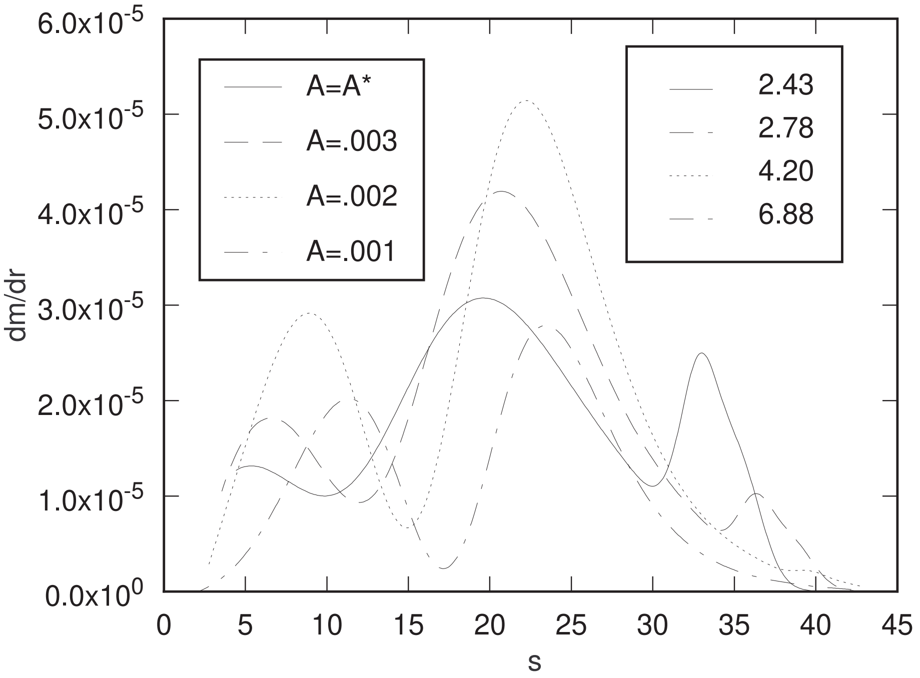

Figure 4.24. $\frac{dM}{dr}$ at $t = 40\,M$ for $\sigma = 2$, $d = 2$, $c = 10$, and various amplitudes (inset shows 1000 times fraction of mass scattered).

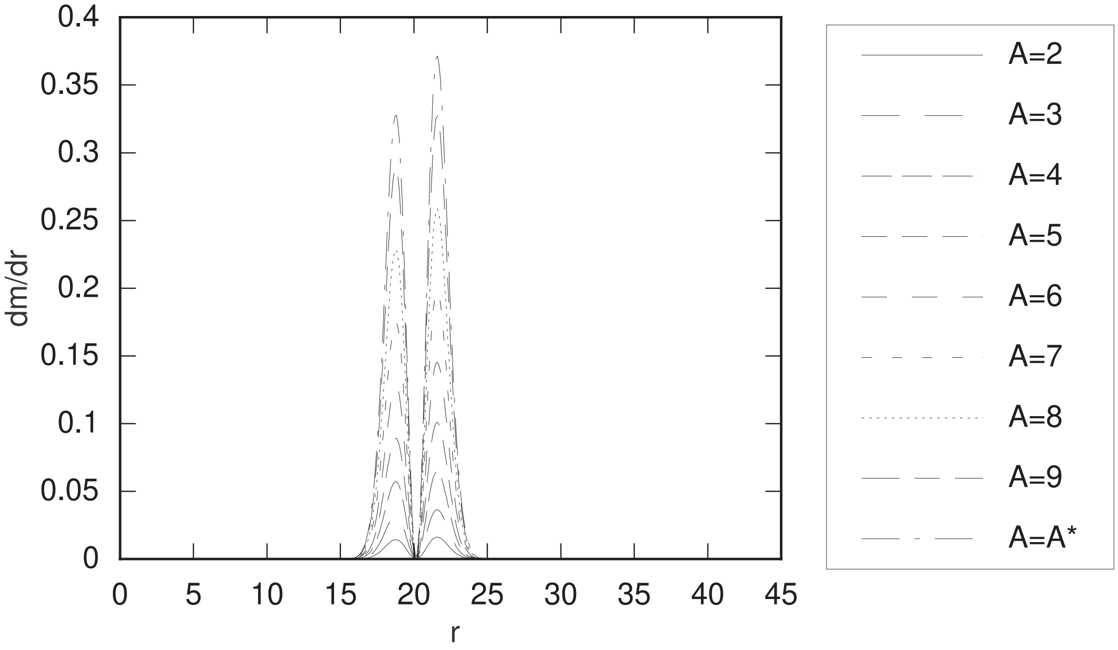

Figure 4.25. $\frac{dM}{dr}$ at $t = 0$ for $\sigma = 2$, $d = 2$, $c = 20$, and various amplitudes ($\times 10^{-4}$).

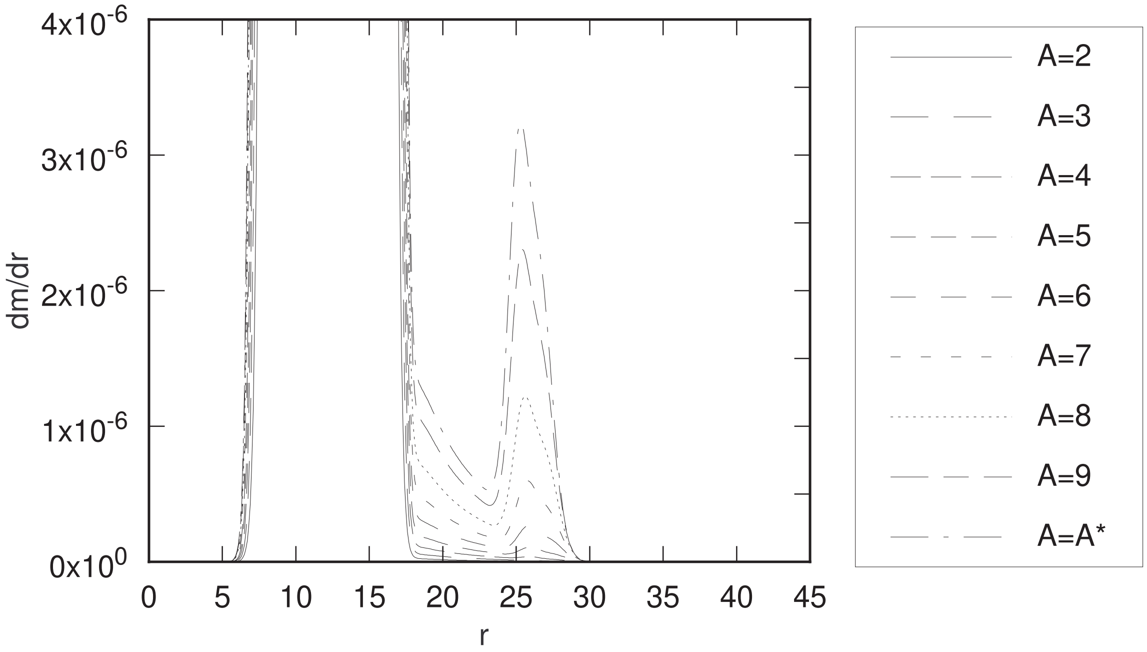

Figure 4.26. $\frac{dM}{dr}$ at $t = 8$ for $\sigma = 2$, $d = 2$, $c = 20$, and various amplitudes ($\times 10^{-4}$).

to tell whether this feature is caused by the nonlinear interaction of an outgoing piece of the initial data with the rest of the pulse, or if it is caused by back-scattering from the effective self-potential of the ingoing pulse. Further study is needed.

4.8. Convergence

The convergence factors for the scalar field and the geometric variables are plotted in Figures 4.27 and 4.28. This evolution is for a sub-critical strong field in which the mass of the scalar field is just over half the mass of the black hole. These factors are approximately four throughout the evolution, indicating that the schemes are second order and convergent (see Section 2.2.2). There are some deviations while the pulse is interacting with the inner boundary due to the forward differencing. Since these derivatives are not centered, their expansions contain terms proportional to odd powers of the grid spacing. These errors can cause grid spacing-dependent phase variations in the reflected pulse which cause the convergence factor to vary during the interaction.

Evolutions of super-critical data exhibit 2nd-order convergence before the scalar field collapses and 1st-order convergence after. This is because evolutions at different resolutions are actually different problems once the new horizon forms. The new horizon will be located at a different radius at each resolution (though the locations will converge as the grid spacing decreases). Thus, the spacetimes contain a final black hole with a resolution-dependent mass. Consider the evolution of super-critical data on two grids, one with a grid spacing of $2h$ and the other with a grid spacing of h. The post-collapse location of the horizon on the high resolution grid will differ from its location on the low resolution grid by h. This is half of the coarse-grid resolution. This difference is precisely enough to cause the convergence rate to drop one order.

Of course the program could be written to demand that the new horizon form on a coarse grid point. However, this would defeat the purpose of performing a higher resolution calculation since it would incur the same error in the black hole mass as the low resolution calculation. This example illustrates that while convergence factors are a valuable tool

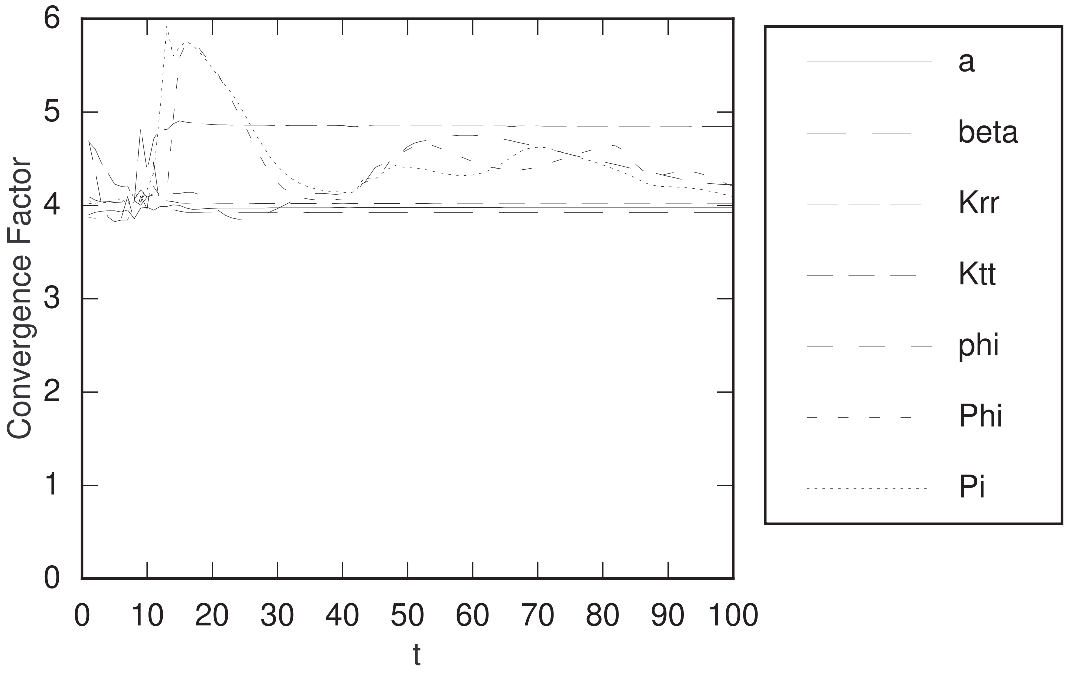

Figure 4.27. Convergence factors for $A = 2.5 \times 10^{-3}$, $c = 10$, and $\sigma = 2$.

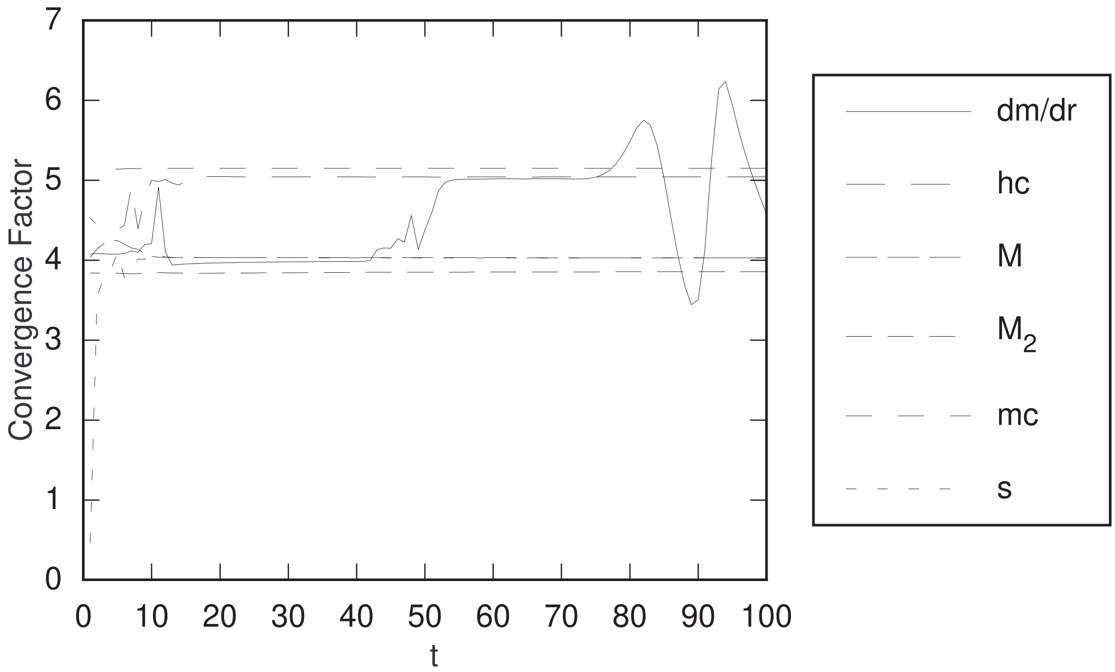

Figure 4.28. Convergence factors for $A = 2.5 \times 10^{-3}$, $c = 10$, and $\sigma = 2$.

for determining the correctness and stability of a finite-difference program, they must be interpreted. A higher rate of convergence does not always mean a more accurate program.

Chapter 5. Spherical Symmetry III: Massive Scalar Field

5.1. A Static Solution

In a recent paper [3], Bechmann and Lechtenfeld investigated the scalar no-hair theorem [4]. This theorem basically states that an asymptotically flat, stationary spacetime containing a black hole is completely specified by the black hole's mass, charge, and angular momentum. There are no other independent parameters. Any initial configurations of matter will be either absorbed into the black hole or radiated to infinity, leaving a Kerr-Newman black hole.

In their paper, Bechmann and Lechtenfeld showed that by weakening one of the conditions of the theorem to allow a locally negative potential, they could construct a static spherically symmetric solution to the Einstein-Klein-Gordon equations which includes a non-trivial scalar field surrounding a black hole—a black hole with scalar "hair."

Unfortunately, their solution is not given in closed form. Thus, any detailed investigation of its stability properties must be carried out numerically. In order to investigate this solution using methods discussed in Chapters 3 and 4, we must perform a coordinate transformation to get the Bechmann and Lechtenfeld solution into the MMIEF coordinate system.

5.2. Coordinate Transformation

The Bechmann and Lechtenfeld solution uses a metric with the following form:

$$ds^2 = -G(\tilde{r})\, d\tilde{t}^2 + G(\tilde{r})^{-1} d\tilde{r}^2 + S(\tilde{r})\, d\tilde{\Omega}^2. \tag{5.2.1}$$

We will transform this using two coordinate transformations, one for the radial coordinate and the other for the time coordinate. The angular coordinates are left unchanged.

First we will transform from $(\tilde{t},\ \tilde{r})$ to $(T,\ R)$ using the transformation

$$\begin{cases} R \equiv S(\tilde{r}) & d\tilde{r} = dR\left(\dfrac{\partial S}{\partial \tilde{r}}\right)^{-1} \\ T \equiv \tilde{t} & d\tilde{t} = dT. \end{cases} \tag{5.2.2}$$

This results in the metric

$$ds^2 = -G\left(\tilde{r}(R)\right)dT^2 + \left(\frac{\partial S}{\partial \tilde{r}}\right)^{-2} G(\tilde{r})^{-1} dR^2 + R^2 d\Omega^2, \tag{5.2.3}$$

where $\tilde{r}$ is assumed to be a function of R. We now transform from $(T,\ R)$ to $(t,\ r)$ using

$$\begin{cases} r \equiv R & dR = dr \\ t \equiv T + F(R) & dT = dt - \dfrac{dF}{dR}dR \equiv dt - f(r)\,dr, \end{cases} \tag{5.2.4}$$

where $f(r)$ is to be determined. The metric becomes

$$ds^2 = -G(r)\,dt^2 + 2\,G(r)f(r)\,dtdr + \left(G(r)^{-1}\left(\frac{\partial S}{\partial \tilde{r}}\right)^{-2} - G(r)f^2(r)\right)dr^2 + r^2 d\Omega^2, \tag{5.2.5}$$

where $G(r) \equiv G\left(\tilde{r}(r)\right)$. Comparing this metric with (3.2.7) gives us the following three equations which must be solved for a, β, and f.

$$\begin{cases} a^2 = G^{-1}\left(\dfrac{\partial S}{\partial \tilde{r}}\right)^{-2} - G f^2(r) \\ 2\,a^2\beta = 2\,Gf \\ a^2(2\beta - 1) = -G. \end{cases} \tag{5.2.6}$$

If we define $Q \equiv G^2\left(\dfrac{\partial S}{\partial \tilde{r}}\right)^2$ then we get the following solutions for a and β:

$$a^2 = \frac{G}{Q}\left(1 - Qf^2\right), \tag{5.2.7}$$

$$\beta = \frac{Qf}{1 - Qf^2}. \tag{5.2.8}$$

This gives us the following simple equation for f :

$$Qf^2 + 2Qf + Q - 1 = 0. \tag{5.2.9}$$

Solving gives

$$f = \pm Q^{-\frac{1}{2}} - 1 = \pm \left(G \frac{\partial S}{\partial \tilde{r}} \right)^{-1} - 1. \tag{5.2.10}$$

We now see that

$$a^2 = -G \pm 2 \left(\frac{\partial S}{\partial \tilde{r}} \right)^{-1}, \tag{5.2.11}$$

and

$$\beta = \frac{-G \left(\frac{\partial S}{\partial \tilde{r}} \right) \pm 1}{-G \left(\frac{\partial S}{\partial \tilde{r}} \right) \pm 2}. \tag{5.2.12}$$

5.3. Initial Data

The Bechmann and Lechtenfeld solution is as follows:

$$\phi(\tilde{r}) = \phi_0 e^{-m\tilde{r}},$$

$$S(\tilde{r}) = \frac{1}{m} \left[K_0 \left(\frac{1}{2} \phi(\tilde{r}) \right) + \left(\ln \left(\frac{\phi_0}{4} \right) + \gamma \right) I_0 \left(\frac{1}{2} \phi(\tilde{r}) \right) \right],$$

$$G(\tilde{r}) = S^2(\tilde{r}) \int_{\tilde{r}}^{\infty} \frac{2r' - 6M}{S^4(r')} dr', \tag{5.3.1}$$

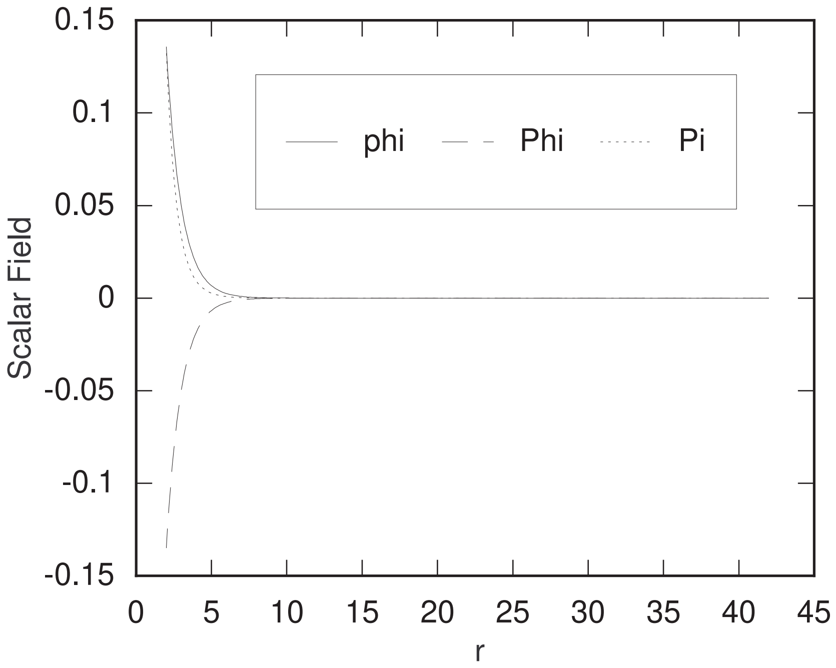

Figure 5.1. Initial data for the scalar field.

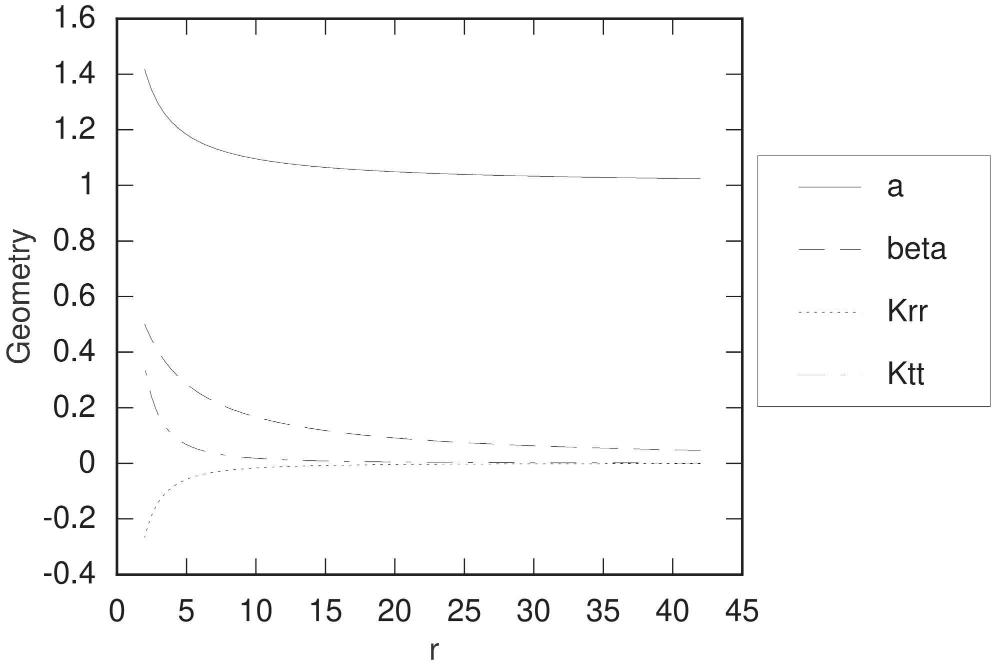

Figure 5.2. Initial data for the geometric variables.

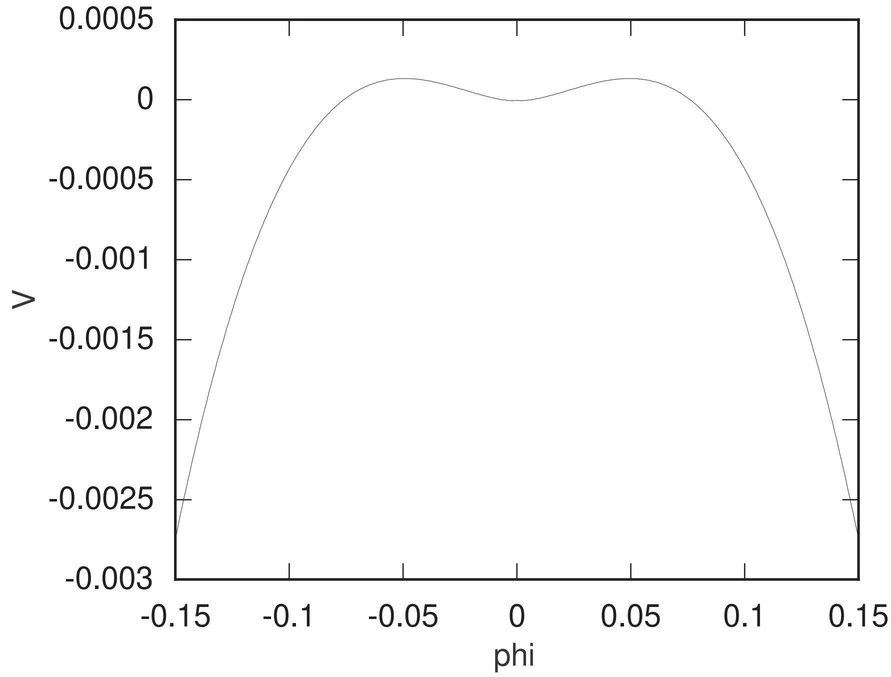

Figure 5.3. The potential $V(\phi)$.

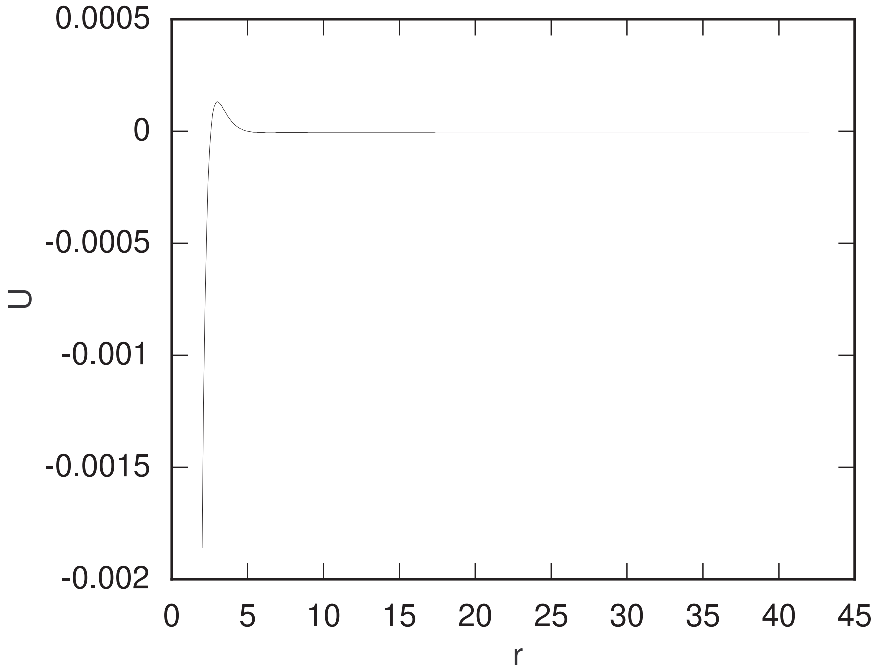

Figure 5.4. The potential $U(r)$.

$$U(\tilde{r}) = \frac{1}{2} G(\tilde{r}) \frac{\left(S^4(\tilde{r})\right)''}{S^4(\tilde{r})} + (2\tilde{r} - 6M)\left(\frac{1}{S^2(\tilde{r})}\right)' - \frac{2}{S^2(\tilde{r})},$$

$$3M = \frac{\int_h^\infty r S^{-4}(r)\, dr}{\int_h^\infty S^{-4}(r)\, dr},$$

where K_0 and I_0 are the *modified Bessel functions* of order zero.

These equations involve four constants, m, ϕ_0, M, and h. We will specify the first three and calculate h from the equation $G(h) = 0$. We will calculate the initial data in the $(r,\ t)$ coordinate system using the following equations.

The scalar field is easily calculated from

$$\phi(r) = \phi\left(\tilde{r}(r)\right) = \phi\left(S^{-1}(r)\right) \tag{5.3.2}$$

The functions Φ and Π are calculated from this and equations (3.1.13) and (3.1.14).

The metric functions a and β are calculated from (5.2.11) and (5.2.12), given that

$$\frac{\partial S}{\partial \tilde{r}} = \frac{1}{2}\phi\left[K_1\left(\frac{1}{2}\phi\right) - \left(\ln\left(\frac{\phi_0}{4}\right) + \gamma\right) I_1\left(\frac{1}{2}\phi\right)\right]. \tag{5.3.3}$$

We can calculate the extrinsic curvature components from (3.1.9) and (3.2.3). We get

$$K^r{}_r = \frac{(a\beta)'}{a^2(1-\beta)}, \tag{5.3.4}$$

$$K^\theta{}_\theta = \frac{\beta - \dot{f}}{ra(1-\beta)}. \tag{5.3.5}$$

The resulting initial data is plotted in Figures 5.1 and 5.2.

5.4. Finite Difference Equations

The equations are discretized using the same methods as in the massless case (see Chapter 4). The potential is handled by introducing two auxiliary variables, U and U_p. These are defined by

$$U(r) \equiv V\Big(\phi(r)\Big) \tag{5.4.1}$$

and

$$U_p(r) \equiv \partial_\phi V\Big(\phi(r)\Big). \tag{5.4.2}$$

The potential and its derivative are discretized in ϕ and are constant throughout the evolution. However, U and U_p will change with time if ϕ changes with time.

Equation (3.2.13) is discretized like equation (4.1.2) with the potential term given by

$$-8\pi A_t U_i^n. \tag{5.4.3}$$

Equation (3.2.16) is discretized like equation (4.1.5) with the added term

$$-A_t\Big(a^2(1-\beta)\,U_p\Big)_i^n. \tag{5.4.4}$$

Equation (3.2.20) is discretized like equation (4.1.6) with the additional factor of

$$\Big(1-8\pi A_t\big(s^2U\big)_i^n\Big) \tag{5.4.5}$$

in the denominator. The auxiliary variables U and U_p are found from V and $\partial_\phi V$ by interpolation. That is, we have

$$U_i^{n+1} = B_i V_j + C_i V_{j+1} \tag{5.4.6}$$

and

$$U_{p_i}^{n+1} = B_i\left(\partial_\phi V\right)_j + C_i\left(\partial_\phi V\right)_{j+1}, \tag{5.4.7}$$

where $j = \left(\phi_i^{n+1} - p_{\min}\right)/\Delta p$, $B_i = 1 - \left(\phi_i^{n+1} - p_i\right)/\Delta p$, $C_i = \left(\phi_i^{n+1} - p_i\right)/\Delta p$, and $p_i \equiv p_{\min} + i\,\Delta\,p$ is the ϕ coordinate on which V and $\partial_\phi V$ are defined.

5.5. Convergence

The convergence of the program can not be demonstrated directly on the Bechmann and Lechtenfeld solution. This solution is a critical solution in the continuum, but not at finite resolution. Thus, discretizations of this solution on two different grids are likely to exhibit different behaviors since they will be different distances from criticality. The solution has a finite lifetime in each case, but "decays" differently depending on the resolution. These different discretizations do not in fact represent the same solution and so will not converge.

In order to test the convergence of the program, we must introduce a perturbation which will move the continuum solution away from criticality. Once this has been done, we should be able to convergence test in the usual manner. The ideal method for the perturbation would be to find two solutions, one super-critical and one sub-critical and interpolate between them using a single parameter. This would allow a search for the critical solution at each discretization. Unfortunately, such a procedure must be left for the future, since it is not obvious how to determine such a family of interpolating solutions for the given problem.

Without such a family, we must resort to choosing an arbitrary perturbation. Consider the following:

$$\phi = \phi_u + \varepsilon \frac{F(z)}{r}, \tag{5.5.1}$$

where $z \equiv r + t$ and ϕ_u is the unperturbed solution. The auxiliary scalar variables are then given by

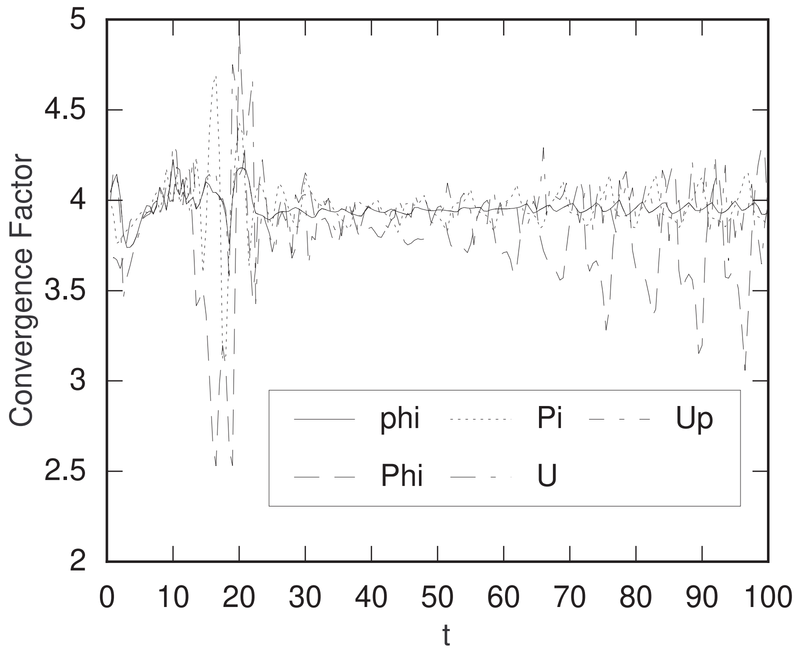

Figure 5.5. Convergence of the scalar field.

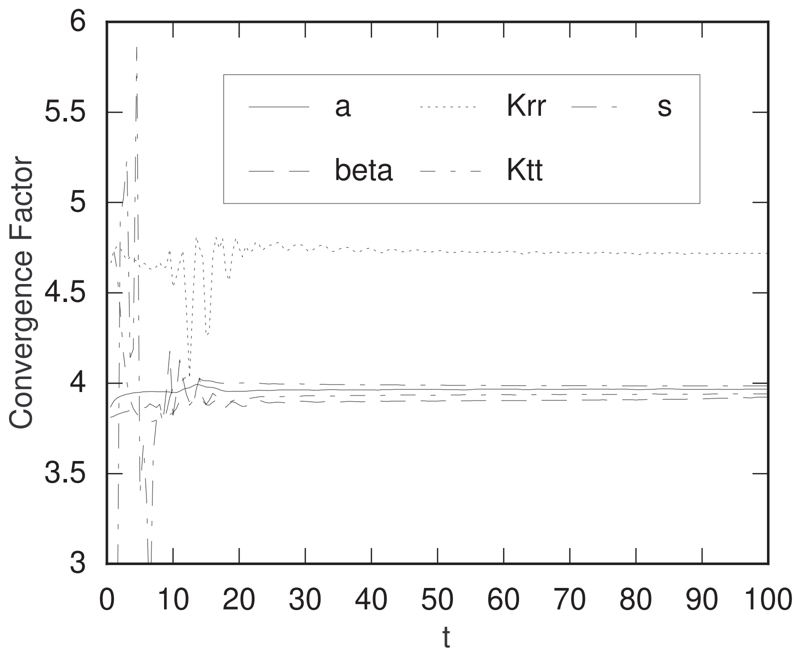

Figure 5.6. Convergence of the geometric variables.

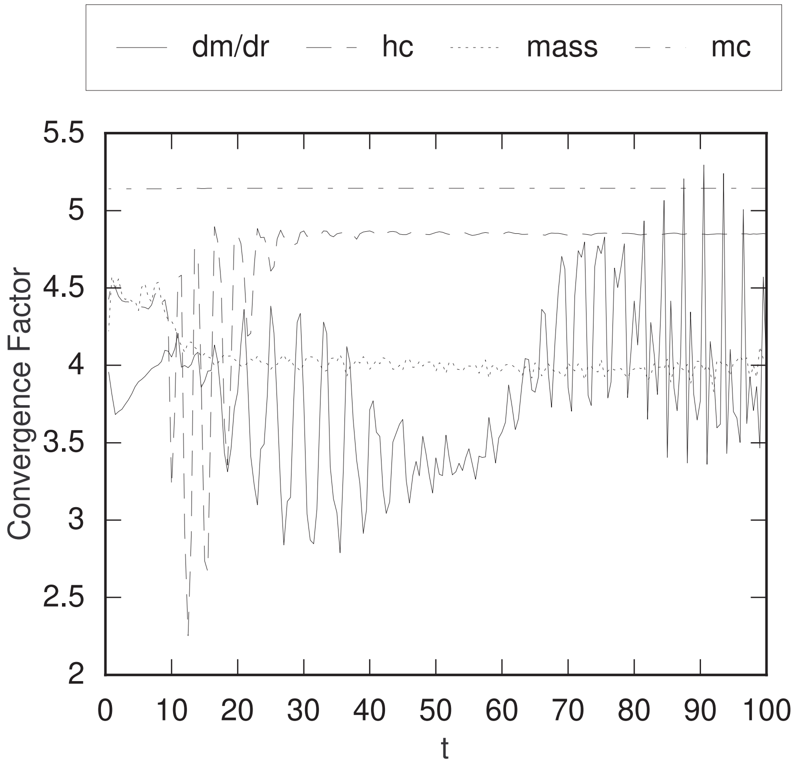

Figure 5.7. Convergence of the derived variables.

$$\Phi = \Phi_u + \varepsilon\left(\frac{\partial_z F}{r} - \frac{F}{r^2}\right) \tag{5.5.2}$$

and

$$\Pi = \Pi_u + \varepsilon\left(\frac{\partial_z F}{r} + \frac{\beta}{1-\beta}\frac{F}{r^2}\right). \tag{5.5.3}$$

We will leave the potential unperturbed, that is, $V = V_u$. For the geometric variables, we will leave ${K^r}_r$ alone and solve for β, ${K^\theta}_\theta$, and a using equations (3.2.5), (3.2.11), and (3.2.10).

Although this perturbation leaves the scalar field variables "close" to the critical solution, it moves the geometric variables a and ${K^\theta}_\theta$ very far from the critical solution. This is another indication of the instability of the critical solution.

As discussed in Section 2.2.2, the convergence factor is specific to the data being evolved. If the program converges on one set of initial data, it will not necessarily converge on another.

However, convergence on one set of strong-field initial data will usually indicate the efficacy of the program for solving similar problems.

In this case there is not much choice. Without further work, there is no way to test convergence on critical or near-critical solutions. Thus, we must test the program on other data. Since the program is designed to allow an arbitrary potential to be given in the initial data (the potential is discretized in ϕ and does not need to be given explicitly in the equations), we can use a different potential entirely. For instance, we can study the massive scalar field by introducing a potential of the form

$$V(\phi) = \frac{1}{2} m^2 \phi^2, \tag{5.5.4}$$

where m represents the mass of the scalar particle. Using this potential along with an ingoing Gaussian pulse (see Section 4.2), we can look at the scattering of the massive scalar pulse. With this data, the program converges to second order as shown in Figures 5.5, 5.6, and 5.7. The convergence of this program on strong field data provides more evidence that the evolutions of the Bechmann and Letchtenfeld solution can be trusted, and hence that this scalar "hair" is unstable.

Chapter 6. Massless Klein-Gordon Equation on a Kerr Background: Theory

6.1. Evolution Equations

An axially symmetric spacetime possesses only a single Killing vector (see [24]). We can choose our coordinate system so that this Killing vector is one of the basis vectors. We then find that the coordinate corresponding to this vector is ignorable, that is, all quantities are functions of the other three coordinates alone. However, the metric is not restricted in form—all components can be non-zero. If the coordinate system is not adapted to the symmetry of the spacetime, then no coordinate is ignorable and we are faced with using the general form of the Einstein equations (see Section 2.1).

In this chapter, we are not concerned with solving the Einstein equations, rather we seek to solve the much simpler linear problem of the massless Klein-Gordon equation on a fixed Kerr Background. Thus, we need only the evolution equations for the scalar field given a general $3+1$ metric. These are the four equations (2.1.12) and (2.1.13).

6.2. Kerr Initial Data

The geometric variables are set to represent an isolated Kerr black hole with mass m and angular momentum parameter a. We adopt Kerr-Schild coordinates $(t,\ x,\ y,\ z)$. The metric in these coordinates is given by

$$g_{\mu\nu} = \eta_{\mu\nu} + H k_\mu k_\nu, \tag{6.2.1}$$

where $\eta_{\mu\nu}$ is the flat-space metric, H is a scalar function, and k^{μ} is tangent to the principle ingoing null congruence. H is given by

$$H = \frac{2\,m\,r^3}{r^4 + a^2 z^2} \tag{6.2.2}$$

and the components of k_μ are given by

$$k_\mu = \left(1, \frac{rx + ay}{r^2 + a^2}, \frac{ry - ax}{r^2 + a^2}, \frac{z}{r}\right), \tag{6.2.3}$$

where

$$r = \sqrt{\frac{x^2 + y^2 + z^2 - a^2}{2} + \sqrt{\left(\frac{x^2 + y^2 + z^2 - a^2}{2}\right)^2 + z^2 a^2}} \tag{6.2.4}$$

and the z axis is the axis of rotation.

The three-metric is given by

$$h_{ij} = \delta_{ij} + H k_i k_j \tag{6.2.5}$$

and its inverse is given by

$$h^{ij} = \frac{1}{1+H}\begin{pmatrix} 1 + H\left({k_y}^2 + {k_z}^2\right) & -Hk_x k_y & -Hk_x k_z \\ -Hk_x k_y & 1 + H\left({k_x}^2 + {k_z}^2\right) & -Hk_y k_z \\ -Hk_x k_z & -Hk_y k_z & 1 + H\left({k_x}^2 + {k_y}^2\right) \end{pmatrix}. \tag{6.2.6}$$

The lapse is

$$\alpha = \sqrt{\frac{1}{1+H}} \tag{6.2.7}$$

and the shift is

$$\beta^i = \frac{H}{1+H} k^i. \tag{6.2.8}$$

The determinant of the three metric is

$$h = 1 + H. \tag{6.2.9}$$

Since I am examining the scattering of scalar radiation from a *fixed* Kerr source, I can simply specify the scalar field and leave the geometry as above. The scalar field will be a compact pulse given by

$$\phi = A \exp\left(-\left(\frac{x-c_x}{\delta_x}\right)^2\right) \exp\left(-\left(\frac{y-c_y}{\delta_y}\right)^2\right) \exp\left(-\left(\frac{z-c_z}{\delta_z}\right)^2\right) \tag{6.2.10}$$

and

$$\dot{\phi} = -2\frac{x-c_x}{\delta_x{}^2} A \exp\left(-\left(\frac{x-c_x}{\delta_x}\right)^2\right) \exp\left(-\left(\frac{y-c_y}{\delta_y}\right)^2\right) \exp\left(-\left(\frac{z-c_z}{\delta_z}\right)^2\right), \tag{6.2.11}$$

or by

$$\phi = A \exp\left(-\left(\frac{r-c}{\delta}\right)^2\right) \cos(n\varphi) \tag{6.2.12}$$

and

$$\phi = -2\left(\frac{r-c}{\delta^2}\right) A \exp\left(-\left(\frac{r-c}{\delta}\right)^2\right) \cos(n\varphi), \tag{6.2.13}$$

where n is an integer (see below).

6.3. Super Radiance

One of the interesting properties of the Kerr black hole is its ability to amplify low-frequency waves (see [9]). We can see this effect most easily if we adopt Boyer-Lindquist coordinates. In these coordinates, the metric is

$$ds^2 = dr^2 - 2\,a\sin^2\theta\,dr d\varphi + \left(r^2 + a^2\right)\sin^2\theta\,d\varphi^2 + \Sigma\,d\theta^2 - dt^2$$

$$+\frac{2mr}{\Sigma}\left(dr - a\sin^2\theta\,d\varphi + dt\right)^2, \tag{6.3.1}$$

where $\Sigma = r^2 + a^2\cos^2\theta$. If we write the scalar field as $\phi = R\left(r\right)P\left(\theta\right)e^{in\varphi-\omega t}$, where n is the azimuthal quantum number and ω is a frequency, then the Klein-Gordon equation is separable. The radial equation is

$$\Delta\frac{d}{dr}\left(\Delta\frac{dR}{dr}\right) + \left[\omega^2\left(r^2 + a^2\right) - 4\,a\,m\,n\,r\,\omega + a^2n^2 - \lambda\Delta\right]R = 0\,, \tag{6.3.2}$$

where $\Delta = r^2 + a^2 - 2\,m\,r$ and λ is a separation constant. If we define a new coordinate z by

$$\frac{dr}{dz} = \frac{\Delta}{r^2 + a^2}\,, \tag{6.3.3}$$

and a new radial function f by

$$f\left(r\right) = R\left(r\right)\sqrt{r^2 + a^2}, \tag{6.3.4}$$

then (6.3.2) becomes

$$\frac{d^2f}{dz^2} + F^2\left(z\right)f = 0\,, \tag{6.3.5}$$

where

$$F^2\left(z\right) = \left(\omega - n\,\omega_c\right)^2 - \frac{\Delta}{\left(r^2 + a^2\right)^2}\left\{\lambda - 2\,a\,n\,\omega + \sqrt{r^2 + a^2}\,\frac{d}{dr}\left[\frac{\Delta\,r}{\left(r^2 + a^2\right)^{3/2}}\right]\right\}, \tag{6.3.6}$$

$$\omega_H = \frac{a}{r_+^2 + a^2}, \tag{6.3.7}$$

and $r_+ = m + \sqrt{m^2 - a^2}$ is the location of the horizon. Now (6.3.2) admits two linearly independent solutions which we can define asymptotically by noting that $F^2(z) \to \omega^2$ as $z \to \infty \ (r \to \infty)$ and $F^2(z) \to (\omega - n\,\omega_H)^2$ as $z \to -\infty \ (r \to r_+)$. Now the solution must be purely ingoing at the horizon, so

$$f \sim e^{-i(\omega - n\omega_H)z}. \tag{6.3.8}$$

At infinity, we get a combination of ingoing and outgoing parts

$$f = A\,e^{-i\omega z} + B\,e^{i\omega z}. \tag{6.3.9}$$

The second solution is given by the complex conjugate of the first. The Wronskian at the inner boundary is

$$\begin{vmatrix} e^{-i(\omega - n\omega_H)z} & e^{i(\omega - n\omega_H)z} \\ -i(\omega - n\,\omega_H)\,e^{-i(\omega - n\omega_H)z} & i(\omega - n\,\omega_H)\,e^{i(\omega - n\omega_H)z} \end{vmatrix} = 2\,i(\omega - n\,\omega_H).$$

At the outer boundary, we have

$$\begin{vmatrix} A\,e^{-i\omega z} + B\,e^{i\omega z} & A^* e^{i\omega z} + B^* e^{-i\omega z} \\ -i\,\omega A\,e^{-i\omega z} + i\,\omega B\,e^{i\omega z} & i\,\omega A^* e^{i\omega z} - i\,\omega B^* e^{-i\omega z} \end{vmatrix} = 2\,i\,\omega(|A|^2 - |B|^2).$$

Since both asymptotic solutions are part of a single solution, it must be true that the Wronskians are equal. Thus we have

$$(\omega - n\,\omega_H) = \omega(|A|^2 - |B|^2). \tag{6.3.10}$$

So the wave is amplified, that is, $|B|^2 > |A|^2$ whenever $0 < \omega < n\,\omega_H$ [17]. In terms of wavelength, we have amplification as long as

$$\lambda > \frac{2\,\pi(r_+^2 + a^2)}{n\,a}. \tag{6.3.11}$$

Chapter 7. Massless Klein-Gordon Equation on a Kerr Background: Numerics

7.1. Finite Difference Equations

Equations (2.1.12) and (2.1.13) are discretized in a similar manner to that used in the spherically symmetric case. I use a two-level implicit scheme with dissipation, however, all the spatial derivatives are centered, there are no angled derivatives. I introduce three additional auxiliary functions (*aux_x*, *aux_y*, and *aux_z*) to allow for easier treatment of the boundaries. These functions are only defined on a single time level. The interior equations are

$$\triangle_t^d \Pi^n_{i,j,k} = aux\_x^n_{i,j,k} + aux\_y^n_{i,j,k} + aux\_z^n_{i,j,k} \tag{7.1.1}$$

$$aux\_x^n_{i,j,k} = \triangle_x \left[\beta^x \Pi + \alpha\sqrt{h}\left(h^{xx}\Phi_x + h^{xy}\Phi_y + h^{xz}\Phi_z\right)\right]^n_{i,j,k} \tag{7.1.2}$$

$$aux\_y^n_{i,j,k} = \triangle_y \left[\beta^y \Pi + \alpha\sqrt{h}\left(h^{xy}\Phi_x + h^{yy}\Phi_y + h^{yz}\Phi_z\right)\right]^n_{i,j,k} \tag{7.1.3}$$

$$aux\_z^n_{i,j,k} = \triangle_z \left[\beta^z \Pi + \alpha\sqrt{h}\left(h^{xz}\Phi_x + h^{yz}\Phi_y + h^{zz}\Phi_z\right)\right]^n_{i,j,k} \tag{7.1.4}$$

$$\triangle_t^d \Phi^{\,n}_{x_{i,j,k}} = \triangle_x \left(\frac{\alpha}{\sqrt{h}}\Pi + \beta^x\Phi_x + \beta^y\Phi_y + \beta^z\Phi_z\right)^n_{i,j,k} \tag{7.1.5}$$

$$\triangle_t^d \Phi^{\,n}_{y_{i,j,k}} = \triangle_y \left(\frac{\alpha}{\sqrt{h}}\Pi + \beta^x\Phi_x + \beta^y\Phi_y + \beta^z\Phi_z\right)^n_{i,j,k} \tag{7.1.6}$$

$$\triangle_t^d \Phi_{z\,i,j,k}^{\;n} = \triangle_z \left(\frac{\alpha}{\sqrt{h}} \Pi + \beta^x \Phi_x + \beta^y \Phi_y + \beta^z \Phi_z \right)_{i,j,k}^{n}. \tag{7.1.7}$$

As in the spherically symmetric case, if the point is next to the boundary, then the dissipative time derivatives are replaced by regular time derivatives.

The interior of the black hole is marked by a characteristic function which is zero inside the horizon and one outside. Because of the auxiliary functions, each evolution equation needs to perform a derivative in only one dimension. This means there are only four boundaries for each equation, one on each side of the hole and one on each edge of the grid. The horizon boundaries are handled in the same way as in the spherically symmetric case, that is, by using the evolution equations with backward derivatives. For instance, the updates for Φ_x on each side of the hole are

$$\triangle_t^d \Phi_{x\,i,j,k}^{\;n} = \triangle_x^f \left(\frac{\alpha}{\sqrt{h}} \Pi + \beta^x \Phi_x + \beta^y \Phi_y + \beta^z \Phi_z \right)_{i,j,k}^{n} \tag{7.1.8}$$

on $+x$ side of the hole and

$$\triangle_t^d \Phi_{x\,i,j,k}^{\;n} = \triangle_x^b \left(\frac{\alpha}{\sqrt{h}} \Pi + \beta^x \Phi_x + \beta^y \Phi_y + \beta^z \Phi_z \right)_{i,j,k}^{n} \tag{7.1.9}$$

on the $-x$ side.

At the edges of the grid, I use outgoing wave boundary conditions. Again, as in the spherically symmetric case, I assume that near the boundaries, the wave is outgoing and spherical, that is

$$r\phi \sim F(r - ct). \tag{7.1.10}$$

Since we have $x^2, y^2, z^2 \gg a^2$, we can assume that $r \approx \sqrt{x^2 + y^2 + z^2}$ and that the shift components are small. The spatial derivatives of F give the following three conditions

$$\partial_t \phi + \frac{\phi}{r} + \frac{r}{x} \partial_x \phi = 0 \tag{7.1.11}$$

$$\partial_t \phi + \frac{\phi}{r} + \frac{r}{y}\partial_y \phi = 0 \tag{7.1.12}$$

$$\partial_t \phi + \frac{\phi}{r} + \frac{r}{z}\partial_z \phi = 0\,. \tag{7.1.13}$$

Differentiating these gives equations for the three Φ variables:

$$\partial_t \Phi_x + \frac{2}{r} - \frac{r}{x^2}\Phi_x + \frac{r}{x}\partial_x \Phi_x - \frac{x}{r^3}\phi = 0 \tag{7.1.14}$$

$$\partial_t \Phi_y + \frac{2}{r} - \frac{r}{y^2}\Phi_y + \frac{r}{y}\partial_y \Phi_y - \frac{y}{r^3}\phi = 0 \tag{7.1.15}$$

$$\partial_t \Phi_z + \frac{2}{r} - \frac{r}{z^2}\Phi_z + \frac{r}{z}\partial_z \Phi_z - \frac{z}{r^3}\phi = 0 \tag{7.1.16}$$

Π can be updated by substituting one of (7.1.11), (7.1.12), or (7.1.13) into (2.1.10), depending on which edge makes up the boundary. For instance, at $y = y_{\max}$ we would use (7.1.12) to get

$$\Pi = -\frac{\sqrt{h}}{\alpha}\left(\frac{\phi}{r} + \beta^x \Phi_x + \left(\frac{r}{y} + \beta^y\right)\Phi_y + \beta^z \Phi_z\right). \tag{7.1.17}$$

These equations are differenced using backward spatial differences like

$$\triangle_t \Phi^{\,n}_{y_{i,j,k}} + \left(\frac{2}{r_{i,j,k}} - \frac{r_{i,j,k}}{y_j^2}\right) A_t \Phi^{\,n}_{y_{i,j,k}} + \frac{r_{i,j,k}}{y_j}\triangle^b_y \Phi^{\,n}_{y_{i,j,k}} + \frac{y_j}{r^3_{i,j,k}} A_t \phi^n_{i,j,k} = 0\,. \tag{7.1.18}$$

7.2. Convergence

The convergence factors for the scalar field are shown in figure 7.1. These convergence factors were calculated from runs on grids of size 17^3, 33^3, and 65^3, with coordinate ranges from -10 to 10, so the resolution was quite low in each case. Given such poor resolution, we can not expect convergence factors of four, since the asymptotic expansions (2.2.1.4) of the

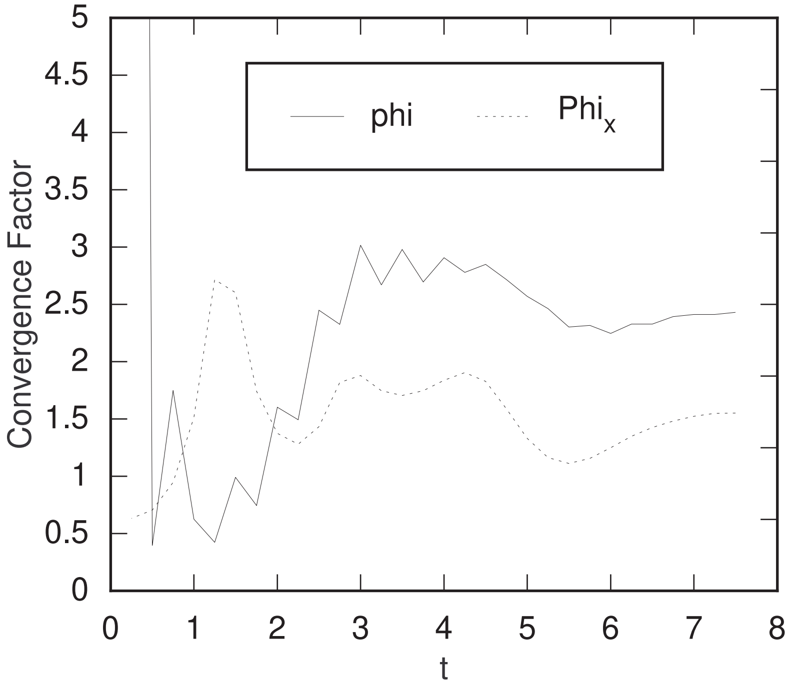

Figure 7.1. Convergence of 3D Scalar Field.

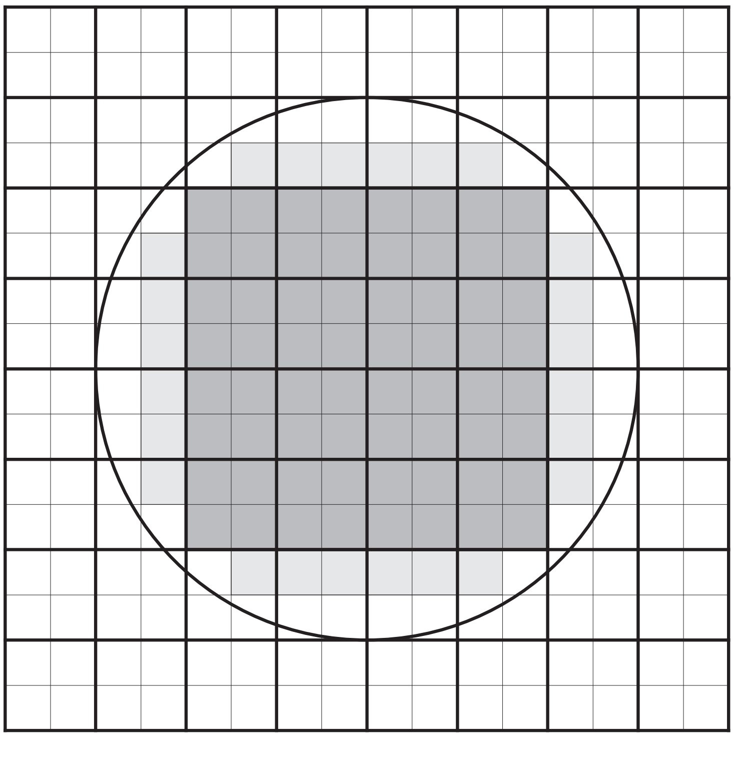

Figure 7.2. Cross-section of the horizon.

grid functions only hold in the limit as the grid spacing shrinks to zero. However, the evidence shown in the figure suggests that the difference scheme is convergent, though it appears that the convergence factor is two.

The 1st-order convergence is caused by the “inner” boundary, that is, the apparent horizon. While the outer boundary of the grid has the same coordinate values at every resolution, the coordinate values of the points in the interior of the black hole change with resolution. Figure 7.2 shows a section through the center of the black hole. The circle represents the actual location of the horizon. The thick lines intersect at the coarse and fine grid points, while the thin lines intersect only at the fine grid points. The dark grey area represents the interior of the black hole on the coarse grid. The light grey area is inside the black hole on the fine grid. Likewise, the “innermost” points outside the black hole change with resolution, though these changes are not marked explicitly in the figure.

As discussed in Section 4.8, differences of half the grid spacing in boundary location will cause the order of convergence to drop. This “problem” could be fixed the same way the super-critical spherically symmetric evolutions could be fixed, but such a “fix” would produce a *less* accurate program.

7.3. Super Radiance

Ideally, to study super radiance, we need a set of monochromatic spherical harmonics to scatter off the black hole. However, such waves are difficult to achieve given the small physical domain and poor resolution available via computer at this time. Given that we cannot get a long-wavelength monochromatic wave, we can use an ingoing gaussian packet with a width much greater than the critical wavelength (6.3.11). For a black hole with $m = 1$, λ_H ranges from ∞ at $a = 0$ to 4π at $a = 1.0$. This means that even for a critical ($m = a$) black hole, λ must be greater than about $12/n$. For a black hole with $a = .5m$ we need λ to be greater than $47/n$. These wavelengths are much too large for the numerical domain when $n = 1$. Obviously, given a large enough n we can get an arbitrarily small critical wavelength. However, the poor resolution not only means the spatial extent of the grid is restricted, but that n can not be too

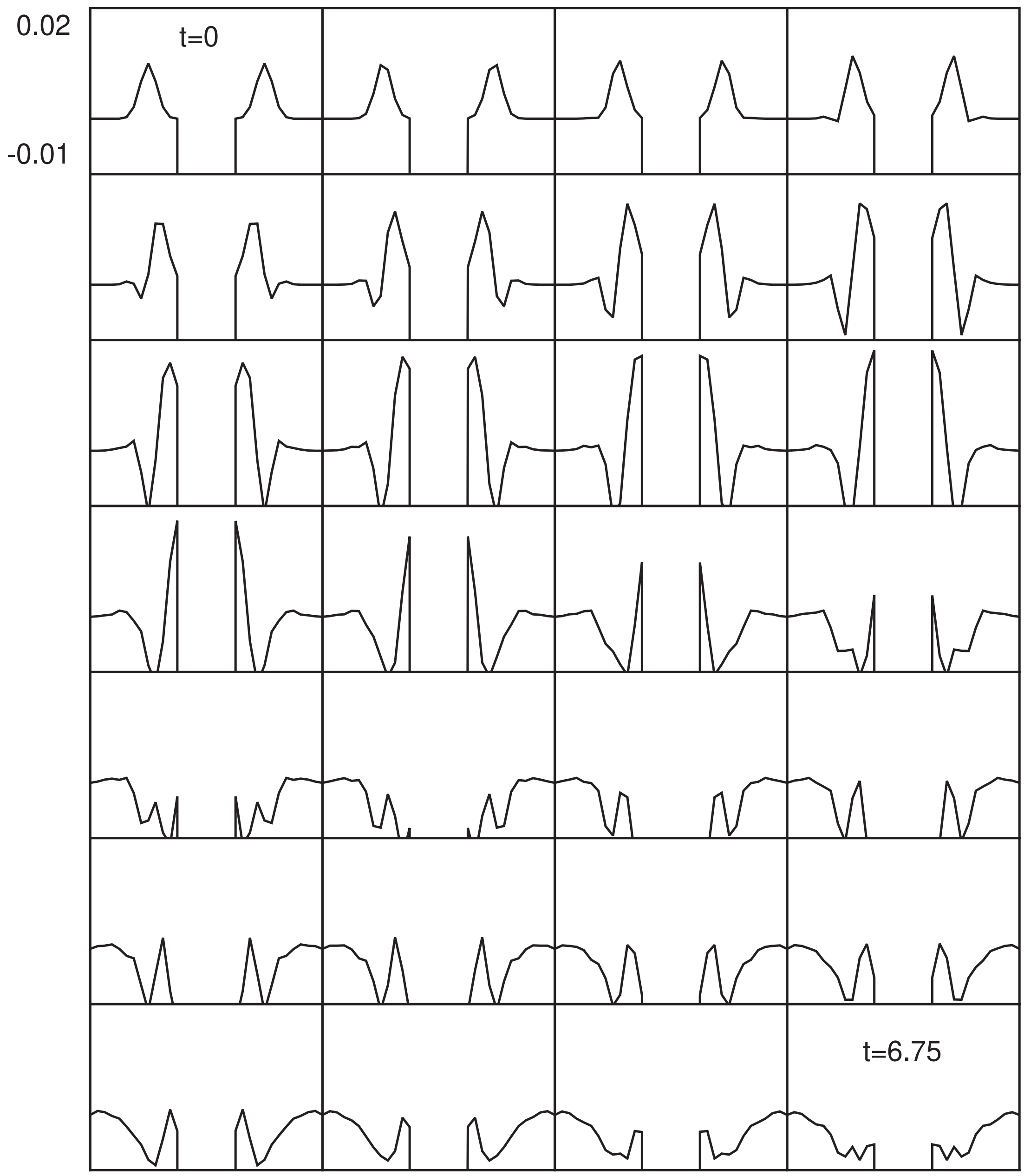

Figure 7.3. Evolution of compact spherical pulse.

large. A large n will cause angular variations which are too fine to be resolved by the grid. As it turns out, it is basically impossible to demonstrate super radiance using this program as it stands. It needs to be modified to support adaptive mesh refinements so that the boundaries of the grid can be extended far enough from the black hole to allow large-wavelength initial data while still being able to resolve the near-black hole interactions. However, we can still observe short-wavelength, small n scattering.

To compute the initial data on a Cartesian grid, we'll need equation (6.2.4) and a definition for φ

$$\varphi \equiv \tan^{-1}\left(\frac{ry - ax}{rx + ay}\right). \tag{7.3.1}$$

Using these equations we can compute initial data for the auxiliary scalar fields:

$$\Phi_i = A\, e^{-\left(\frac{r-c}{\delta}\right)^2} \left[\frac{-2\left(r-c\right)}{\delta^2} \frac{\partial r}{\partial x^i} \cos(n\varphi) - n \sin(n\varphi) \frac{\partial \varphi}{\partial x^i} \right], \tag{7.3.2}$$

where

$$\frac{\partial r}{\partial x} = \frac{x}{2r} \left[1 + \frac{x^2 + y^2 + z^2 - a^2}{\left[\left(\frac{x^2 + y^2 + z^2 - a^2}{2} \right)^2 + z^2 a^2 \right]^{½}} \right],$$

$$\frac{\partial r}{\partial y} = \frac{y}{2r} \left[1 + \frac{x^2 + y^2 + z^2 - a^2}{\left[\left(\frac{x^2 + y^2 + z^2 - a^2}{2} \right)^2 + z^2 a^2 \right]^{½}} \right],$$

$$\frac{\partial r}{\partial z} = \frac{z}{2r} \left[1 + \frac{x^2 + y^2 + z^2}{\left[\left(\frac{x^2 + y^2 + z^2 - a^2}{2} \right)^2 + z^2 a^2 \right]^{½}} \right],$$

$$\frac{\partial \varphi}{\partial x} = -\frac{y}{x^2 + y^2} + \frac{a}{r^2 + a^2} \frac{\partial r}{\partial x},$$

$$\frac{\partial \varphi}{\partial y} = \frac{x}{x^2 + y^2} + \frac{a}{r^2 + a^2} \frac{\partial r}{\partial x},$$

and

$$\frac{\partial \varphi}{\partial z} = 0.$$

Figure 7.3 shows the evolution of a compact spherical pulse (see (6.2.12)). The pulse is centered at $r = 6$, has a width of $\delta = 1$, and an azimuthal quantum number of $n = 0$. The figure shows ϕ verses x for $y = z = 0$. As expected, the pulse moves in radially, encounters the hole, falls in, and leaves behind a small outgoing reflection.

The figure also clearly shows the poor resolution and small spatial domain. A proper study of wave phenomena on a Kerr background in Cartesian coordinates is not possible without an adaptive mesh refinement algorithm.

Chapter 8. RNPL: The Language

RNPL (Rapid Numerical Prototyping Language) is a language for expressing time dependent systems of partial differential equations and the finite difference methods used to solve them. It was written specifically with the general relativistic evolution problem in mind, but it can also be used to solve a wide variety of differential systems. The language hides many of the details of a complete solver while still allowing enough freedom to express most finite difference techniques. It is based heavily on ideas developed by Matthew Choptuik throughout his work in numerical relativity.

RNPL was designed to provide the following capabilities:

- equation expression using a "natural" notation
- easy operator expression
- support for a wide range of difference techniques
- easy interfacing with existing programs
- automatic memory management
- check-pointing
- interactive output control
- parameter management
- easy adaptivization
- easy parallelization
- extensibility

Operators	Associativity
^ **	right
- + (unary)	right
* /	left
- +	left
> < >= <=	non
== !=	non
&&	left
||	left
=	non

Table 8.1. RNPL operators in order of precedence

- short development time
- easy changing of finite difference methods
- intelligent defaults

RNPL currently provides all these capabilities except adaptivization and parallelization, both of which will be added shortly. Once these features are added, any existing RNPL program can make use of them with a simple recompilation.

8.1. Program Structure

An RNPL program consists of a series of object declarations. RNPL is strongly typed, so all data objects which are referenced must be declared. There are some exceptions to this rule having to do with coordinates (see Section 9.1). Since RNPL is strictly declarative, there are no "executable" statements. Thus, there is no notion of order as in a traditional programming language. Declarations can occur in any order in the source file. The complete RNPL grammar is shown in Figure 8.1.

RNPL is case-sensitive in general, although case will be ignored if the backend language is case insensitive (see Chapter 9). RNPL statements are made up of tokens which fall into

RNPL Grammar

dec_list	→	
	→	*dec_list declaration*
declaration	→	*param_dec*
	→	*coord_dec*
	→	*grid_dec*
	→	*gfunc_dec*
	→	*attrib_dec*
	→	*d_operator*
	→	*residual*
	→	*initialization*
	→	*looper*
	→	*update*
param_dec	→	**param** *p_type name*
	→	**param** *p_type name becomes scalar*
	→	**param** *p_type name v_size*
	→	**param** *p_type name v_size becomes vector*
	→	**param ivec** *name*
	→	**param ivec** *name becomes ivec_list*
	→	**const param** *p_type name*
	→	**const param** *p_type name becomes scalar*
	→	**const param** *p_type name v_size*
	→	**const param** *p_type name v_size becomes vector*
	→	**sys param** *p_type name becomes scalar*
coord_dec	→	*name* **coordinates** *coord_list*
grid_dec	→	*g_type name* **grid** *name i_region c_region*
	→	*g_type name* **grid** *name*
	→	*g_type name* **obrack** *coord_list* **cbrack grid** *name i_region c_region*
	→	*g_type name* **obrack** *coord_list* **cbrack grid** *name*
gfunc_dec	→	*type name* **on** *name*
	→	*type name* **on** *name* **str**
	→	*type name* **on** *name* **at** *o_list*
	→	*type name* **on** *name* **at** *o_list* **alias**
	→	*type name* **on** *name* **at** *o_list* **str**
	→	*type name* **on** *name* **at** *o_list* **alias str**
attrib_dec	→	**attrib** *p_type name encoding*
	→	**attrib** *p_type name encoding becomes vector*
d_operator	→	**operator** *d_op becomes expr*
residual	→	**resid** *name* **obrace** *res_list opcolon* **cbrace**
	→	**resid** *time index name* **obrace** *res_list opcolon* **cbrace**
residual	→	**evaluate resid** *name* **obrace** *res_list opcolon* **cbrace**
	→	**evaluate resid** *time index name* **obrace** *res_list opcolon* **cbrace**
initialization	→	**initialize** *name* **obrace** *res_list opcolon* **cbrace**
looper	→	**looper** *name*
update	→	*name name* **update** *coord_list* **header** *ref_list*

	$\rightarrow$	**stub** *name* **update** *coord_list* **header** *ref_list*
	$\rightarrow$	**auto** *name* **update** *coord_list*
p_type	$\rightarrow$	**int**
	$\rightarrow$	**float**
	$\rightarrow$	**string**
name	$\rightarrow$	**iden**
scalar	$\rightarrow$	**inum**
	$\rightarrow$	**minus inum**
	$\rightarrow$	**num**
	$\rightarrow$	**minus num**
	$\rightarrow$	**str**
v_size	$\rightarrow$	**obrack inum cbrack**
becomes	$\rightarrow$	**assignop**
	$\rightarrow$	**equals**
vector	$\rightarrow$	**obrack** *scalar_list* **cbrack**
ivec_list	$\rightarrow$	*ivel* **minus** *ivel*
	$\rightarrow$	*ivel* **minus** *ivel* **divide inum**
	$\rightarrow$	*ivec_list* **comma** *ivel* **minus** *ivel*
	$\rightarrow$	*ivec_list* **comma** *ivel* **minus** *ivel* **divide inum**
coord_list	$\rightarrow$	*name*
	$\rightarrow$	*coord_list* **comma** *name*
g_type	$\rightarrow$	**uniform**
	$\rightarrow$	**nonuniform**
i_region	$\rightarrow$	**obrack** *expr* **colon** *expr* **cbrack**
	$\rightarrow$	**obrack** *expr* **colon** *expr* **cbrack** *i_region*
	$\rightarrow$	**obrack** *expr* **colon** *expr* **colon inum cbrack**
	$\rightarrow$	**obrack** *expr* **colon** *expr* **colon minus inum cbrack**
	$\rightarrow$	**obrack** *expr* **colon** *expr* **colon inum cbrack** *i_region*
	$\rightarrow$	**obrack** *expr* **colon** *expr* **colon minus inum cbrack** *i_region*
c_region	$\rightarrow$	**obrace** *name* **colon** *name* **cbrace**
	$\rightarrow$	**obrace** *name* **colon** *name* **cbrace** *c_region*
type	$\rightarrow$	**int**
	$\rightarrow$	**float**
o_list	$\rightarrow$	**inum**
	$\rightarrow$	**minus inum**
	$\rightarrow$	*o_list* **comma inum**
	$\rightarrow$	*o_list* **comma minus inum**
encoding	$\rightarrow$	**encodeone**
	$\rightarrow$	**encodeall**
d_op	$\rightarrow$	*name* **oparen** *expr* **comma** *coord_list* **cparen**
	$\rightarrow$	**expand** *name* **oparen** *expr* **comma** *coord_list* **cparen**

expr	→	*expr* **plus** *expr*
	→	*expr* **minus** *expr*
	→	*expr* **equals** *expr*
	→	*expr* **times** *expr*
	→	*expr* **divide** *expr*
	→	*expr* **caret** *expr*
	→	**plus** *expr*
	→	**minus** *expr*
	→	*expr* **equiv** *expr*
	→	*expr* **noteq** *expr*
	→	*expr* **less** *expr*
	→	*expr* **great** *expr*
	→	*expr* **lesseq** *expr*
	→	*expr* **greateq** *expr*
	→	*expr* **and** *expr*
	→	*expr* **or** *expr*
	→	**oparen** *expr* **cparen**
	→	*d_op*
	→	*func*
	→	*gfunc*
	→	*coord*
	→	*name*
	→	**num**
	→	**inum**
res_list	→	*i_region becomes expr*
	→	*res_list* **scolon** *i_region becomes expr*
	→	*i_region becomes ifstat*
	→	*res_list* **scolon** *i_region becomes ifstat*
opcolon	→	
	→	**scolon**
time	→	**time**
index	→	**obrack inum cbrack**
	→	**obrack minus inum cbrack**
	→	**obrack inum cbrack** *index*
	→	**obrack minus inum cbrack** *index*
ref_list	→	*reference*
	→	ref_list **comma** *reference*
scalar_list	→	*scalar*
	→	*scalar_list scalar*
ivel	→	**inum**
	→	**times**
func	→	*name* **oparen** *expr* **cparen**
gfunc	→	*time name mindex*
coord	→	*name indel*

ifstat	$\rightarrow$	**if** *expr* **then** *expr*
	$\rightarrow$	**if** *expr* **then** *expr* **else** *expr*
	$\rightarrow$	**if** *expr* **then** *expr* **else** *ifstat*
reference	$\rightarrow$	*name*
	$\rightarrow$	*name* **obrack** *coord_list* **cbrack**
	$\rightarrow$	**auto work pound inum oparen** *expr* **cparen**
	$\rightarrow$	**static work pound inum oparen** *expr* **cparen**
mindex	$\rightarrow$	*indel*
	$\rightarrow$	*mindex indel*
indel	$\rightarrow$	**obrack inum cbrack**
	$\rightarrow$	**obrack minus inum cbrack**
	$\rightarrow$	**obrace** *expr* **cbrace**

Figure 8.1. RNPL Grammar

four classes—reserved words, names, operators, and punctuation. These tokens are listed in Table 8.2 and the operators are listed again in Table 8.1. White space (space, tab, newline) is meaningless except as a token separator, so it can be used freely in a source file to provide clarity.

There are two kinds of comments in RNPL programs. The first kind is like a UNIX shell comment. It starts with a `#` at the beginning of a line and continues till the end of the line. The second kind is like a C++ comment. It starts with `//` and ends at the end of the line. The following example illustrates both kinds of comments.

```
# This is the first kind of comment
float A on grid1  // This is the second kind of comment
// So is this
```

8.1.1. Data Objects

There are five kinds of data objects available in an RNPL program: parameters, coordinates, grids, grid functions, and attributes. These objects are in turn made up of scalars, vectors, and index vectors. Scalars are made up of a single integer, float, or string, while vectors are one dimensional arrays of scalars. Index vectors are arrays of triples. The first element of the triple gives a starting index, the second gives an ending index, and the third gives a stride.

Name	Value(s)	Category
param	parameter or PARAMETER	reserved word
ivec	ivec or IVEC	reserved word
constant	constant or CONSTANT	reserved word
sys	system or SYSTEM	reserved word
coordinates	coordinates or COORDINATES	reserved word
grid	grid or GRID	reserved word
obrack	[	punctuation
cbrack	]	punctuation
on	on or ON	reserved word
str	"any characters"	name
at	at or AT	reserved word
alias	alias or ALIAS	reserved word
attrib	attribute or ATTRIBUTE	reserved word
operator	operator or OPERATOR	reserved word
resid	residual or RESIDUAL	reserved word
obrace	{	punctuation
cbrace	}	punctuation
evaluate	evaluate or EVALUATE	reserved word
initialize	initialize or INITIALIZE	reserved word
looper	looper or LOOPER	reserved word
update	update or UPDATE or updates or UPDATES	reserved word
header	header or HEADER	reserved word
stub	stub or STUB	reserved word
auto	auto or AUTO	reserved word
int	int or INT	reserved word
float	float or FLOAT	reserved word
string	string or STRING	reserved word
iden	[a-zA-Z_][a-zA-Z_0-9]*	name
inum	positive integer	name
minus	-	operator, punctuation
num	positive real number	name
assignop	:=	punctuation

equals	=	operator
comma	,	punctuation
divide	/	operator, punctuation
uniform	uniform or UNIFORM	reserved word
nonuniform	nonuniform or NONUNIFORM	reserved word
colon	:	punctuation
encodeone	encodeone or ENCODEONE	reserved word
encodeall	encodeall or ENCODEALL	reserved word
oparen	(	punctuation
cparen	)	punctuation
expand	expand or EXPAND	reserved word
plus	+	operator
times	*	operator, name
caret	^ or **	operator
equiv	==	operator
noteq	!=	operator
less	<	operator
great	>	operator
lesseq	<=	operator
greateq	>=	operator
and	&&	operator
or	||	operator
scolon	;	punctuation
time	<inum> or <-inum>	name
if	if or IF	reserved word
then	then or THEN	reserved word
else	else or ELSE	reserved word
work	work or WORK	reserved word
pound	#	punctuation
static	static or STATIC	reserved word

Table 8.2. RNPL Tokens

8.1.1.1. Parameters

Parameters are constants which are specified at run time. They can be scalars, vectors, or index vectors of any type. They are used to specify things like initial data, grid sizes, and output. Here are some example parameter declarations:

```
parameter int fred

constant parameter float jim := 5

parameter float george[10]

parameter float ted[3] := [2.0 1.7 11]

parameter string name := "file_name"

constant parameter string dft[2] := ["comment 1" "comment 2"]

parameter ivec output := *-10,11-50/10,51-*/15
```

As the examples show, a parameter declaration starts with the reserved word `parameter` or the pair of reserved words `constant parameter`. Then comes a type and a name. Parameters can be assigned default values. An RNPL generated program reads a user specified parameter file on startup. If a parameter has been declared without a default value, it must have a value in the file. The reserved word `constant` is meaningless except when the backend language is FORTRAN. There is also a special type of parameter known as the *system* parameter. System parameters only have meaning when the backend language is FORTRAN (see Section 9.2).

The size of a vector parameter must be included in the declaration (see the declarations for `george`, `ted`, and `dft` above). Scalar parameter declarations contain no size.

An ivec is an index vector. The only current use for index vectors is for controlling output. As shown in the example, the index vector can have a default value specified as a comma-separated list of triples. The third element of each triple (the stride) is optional. If it is not given, it is assumed to be one. The first and second elements can be integers or an asterisk. As the first element, an asterisk means *the first time step.* As the second element, an asterisk means *the last time step.* The example declaration above is interpreted to mean "output every time step from the first time step to the tenth time step, every tenth time step from the eleventh to the fiftieth time step, and every fifteenth time step from the fifty-first to the last time step."

8.1.1.2. Coordinates

AN RNPL program can use multiple coordinate systems. A single time coordinate can be used in multiple coordinate systems, but each spatial coordinate must be unique. Some example coordinate declarations are:

```
rect coordinates t,x,y,z
sph coordinates t,r,theta,phi
null coordinates u,v
```

The first coordinate in a list is assumed to be the time coordinate. The first time coordinate is used for *computational* time. The declaration begins with a user-chosen name, followed by `coordinates`, followed by a list of coordinate names.

8.1.1.3. Grids

Grids define the spatial regions over which the grid functions will be defined as well as their storage. A grid declaration can take one of several forms, the longest of which would be something like:

```
uniform rect[x,z] grid g1 [1:Nx][1:Nz] {xmin:xmax}{zmin:zmax}
```

Grids can be `uniform` or `nonuniform`, though only the former is currently defined. Next comes the name of the coordinate system followed by a list of coordinates on which the grid is defined. The above grid is two dimensional with coordinates x and z. After the coordinate system comes the reserved word `grid` followed by the grid name. Next comes the index region. In this example, the first index starts at 1 and goes to `Nx`, while the second starts at 1 and goes to `Nz`. `Nx` and `Nz` are names of grid sizes. The index regions can contain arbitrary expressions such as `[A*B+C-2:4*Nx-5/a]`, however, as discussed in Section 9.1.1, it is best to keep to forms like `[1:Nx]` and `[0:Nx-1]`. Finally, comes the coordinate region which gives the actual spatial ranges of the coordinates. In the example, we have xmin $\leq x \leq$ xmax and zmin $\leq z \leq$ zmax . Coordinate regions must be of the form `{name1:name2}`, where `name1` and `name2` have been declared as parameters.

Other forms of the grid declaration leave out one or more of the above parts. The minimum allowable declaration is:

```
uniform rect grid g2
```

This declaration (along with the example coordinate declaration in Section 8.1.1.2) declares `g2` to be a three dimensional grid with coordinates x, y, and z. The index region will be `[0:Nx-1][0:Ny-1][0:Nz-1]` for C output and `[1:Nx][1:Ny][1:Nz]` for FORTRAN output. The coordinate region will be `{xmin:xmax}{ymin:ymax}{zmin:zmax}`.

8.1.1.4. Grid Functions

A grid function is a function defined on a grid at one or more times. Some examples of grid function declarations are:

```
float A on g1 at -1,0,1
int B on g2 at 0,1
float C on g1
float D on g2 at -1,0,1 alias
float E on g3 at 0,1 "Electric Field"
```

First comes the grid function type, either `float` or `int`. Next comes the name followed by the reserved word `on` and the grid name on which the function is defined. If the declaration stopped here (such as that for `C` above), we get a single time level. Adding the reserved word `at` followed by a list of offsets (positive or negative integers) gives a function defined on one time level for each offset. For instance the definition for `A` would give a three time level function defined at times $n-1$, n, and $n+1$. Next comes the optional reserved word `alias` which declares common storage for the first and last time levels. Following any of these declarations can be a string which is used as a *print name* for the grid function. There are not any real uses for the print name. It simply provides a more descriptive name by which the grid function will be identified during output and visualization (see Chapter 9 for information on these compiler features).

8.1.1.5. Attributes

An attribute is a flag array associated with the grid functions. For instance, an attribute may tell which grid functions are to be output and which are not. Attributes are defined in a similar manner to vector parameters, except the size is replaced by an encoding. The encoding is either `encodeone` or `encodeall`, with `encodeone` giving one value per grid function and `encodeall` giving one value per time level per grid function (see Section 8.1.1.4 for information about defining grid functions). For instance, if five grid functions are defined, three of which have three time levels each while the remaining two have two levels each, then an attribute marked as `encodeone` would have a length of five, while an attribute marked as `encodeall` would have a length of thirteen. In this case an output flag array could be defined as either

```
attribute int out_gf encodeone
```

or

```
attribute int out_gf encodeone := [0 0 1 1 0]
```

8.1.2. Difference Equations

The “heart” of any RNPL program is the definition of the system of equations to be solved and the methods to be used in solving it. This information is declared using derivative operators, residuals, initializations, and updates.

8.1.2.1. Derivative Operators

Derivative operators are operators which act on grid functions. They are used for turning differential equations into finite difference equations by substituting for continuum differential operators. Here is a declaration for a forward difference operator:

```
operator D_FW(f,r) := (<0>f[1] - <0>f[0])/dr
```

Whenever an operator is used in an expression (see Section 8.1.3), it is replaced by its definition. The name f is arbitrary. It simply shows where the expression goes in the definition. For instance, if D_FW(3*A+B,r) appeared in an expression, the f's in the right hand side would be replaced by 3*A+B. The notation <0>f[1] is interpreted as f^n_{i+1}. The <>[] is really an operator which acts on expressions as follows:

<a>f[b] $\rightarrow$ f if f is a number or parameter or time coordinate

<a>f[b] $\rightarrow f_{i+b}$ if f is a spatial coordinate

<a>f[b] $\rightarrow f^{n+a}_{i+b}$ if f is a grid function

Three dimensional forward difference operators would look like this:

```
operator D_FW(f,x) := (<0>f[1][0][0] - <0>f[0][0][0])/dx
operator D_FW(f,y) := (<0>f[0][1][0] - <0>f[0][0][0])/dy
operator D_FW(f,z) := (<0>f[0][0][1] - <0>f[0][0][0])/dz
```

Operator definitions can be nested as in:

```
operator D_FW(f,r) := (<0>f[1] - <0>f[0])/dr
operator D_BW(f,r) := (<0>f[0] - <0>f[-1])/dr
operator D_CN1(f,r,r) := D_BW(D_FW(<0>f[0],r),r)
operator D_CN2(f,r,r) := D_BW(D_FW(<1>f[0],r),r)
```

As you can predict, the definition of D_CN1 will result in the usual centered second derivative, namely $\left(f^n_{i+1} - 2f^n_i + f^n_{i-1}\right)/dr^2$, while the definition of D_CN2 will result in the same thing applied at the advanced time level, that is $\left(f^{n+1}_{i+1} - 2f^{n+1}_i + f^{n+1}_{i-1}\right)/dr^2$. The list of coordinate names after the f signifies with respect to which coordinate(s) the derivative is taken.

Although operators are defined like derivatives and act as derivatives under certain circumstances (see Section 8.1.3), they can be defined to perform other functions such as the spatial averaging operator defined below.

```
operator AVG(f,r) := (<0>f[1] + <0>f[0])/2
```

Because operators are internally treated as derivatives, even definitions such as this need the coordinate list.

8.1.2.2. Residuals

Residuals define the system of equations, typically by using derivative operators. Consider the following residual definition:

```
residual phi { [0:0]       := D_LF(phi,t) ;
               [1:Nx-2]    := D_LF(phi,t,t) - D_LF(phi,x,x) ;
               [Nx-1:Nx-1] := D_LF(phi,t) }
```

Assuming the proper definitions of the derivative operators, this residual encodes the linear wave equation on a string with the end points fixed. An equivalent declaration would be:

```
residual phi { [0:0]       := D_LF(phi,t) = 0 ;
               [1:Nx-2]    := D_LF(phi,t,t) = D_LF(phi,x,x) ;
               [Nx-1:Nx-1] := D_LF(phi,t) = 0 }
```

First comes the reserved word `residual` followed by the name of the grid function whose residual is being defined. Next comes a bracket-enclosed set of index regions and expressions. The index region shows over what range the expression is a valid description of the behavior of the system. The union of the index regions should equal the index region of the grid on which the function is defined, but this is not required. Note that each region-expression pair is separated by a semicolon. A semicolon can also follow the last expression but is not required.

An alternate form for an index region is `[expr1:expr2:stride]`, where `expr1` and `expr2` are the region bounds as above, and `stride` is an integer stride which can be positive or negative. In the first form, the stride is taken to be one.

The residual tells how to determine the advanced value of a grid function. Thus, the residual must contain `<a>f...` where `a` is the offset to the most advanced time level defined for grid function `f`.

Part of a residual for a three dimensional grid function would look like:

```
residual A { [1:Nx][1:1][1:1] := D_LF(A,x) + D_FW(B,y) ;
             [Nx:Nx][1:Ny][1:Nz] := <1>A[0][0][0] = 5.0*C }
```

The reserved word `residual` can be preceded by the reserved word `evaluate` which tells the compiler to produce code which will evaluate the residual.

In addition, the word `residual` can be followed by a global offset for example:

```
residual <1>[0] A { [1:Nx] := ... }
```

This offset is applied globally to each expression appearing in the residual.

Residuals also support the if-then-else construct. Consider the following declaration:

```
residual A { [1:Nx] := if(<0>C[0] == 1 && <0>C[-1] == 1) then
                         D_LF(A,t) = D_LF(A,x)
                       else if(C == 1) then
                         D_LF(A,t) = D_FW(A,x)
                       else D_LF(A,t) = 0 }
```

Here `C` is a characteristic function which tells which equation belongs in which region.

8.1.2.3. Initializations

An initialization defines the initial data for a grid function. Its form is identical to the residual declaration with `residual` replaced by `initialize`. However the expression is interpreted differently. Consider the following initialization declaration:

```
initialize phi { [1:Nr] := amp*exp(-((r-c)/delta)^2) }
```

Unlike the residual declaration, the expression in the initialization must not contain its grid function (though it can contain other grid functions). The retarded time level of the grid function is set to the expression. In the case above, `phi` will be set to a Gaussian. Initializations are evaluated in the order in which they are defined. Thus, if one grid function is used in the initialization for another, it must be initialized first.

8.1.2.4. Updates

Update declarations can take many forms. Here are some examples:

```
auto update phi,pi,beta
stub evolver updates A,B,C
  header A, B[Bnp1,Bn,Bnm1], C[C], x,y,z,dt,
         auto work#0(5*Nx*Ny*Nz),
         static work#1(3*Nx*Ny-.5*Nz)
myroutine.inc myupdate update A,B header A,B,dt
```

The first form defines an automatic update. The declaration above would cause the compiler to produce a routine to update the grid functions `phi`, `pi`, and `beta` if residuals have been declared for them. Otherwise, the compiler has no idea how to update the grid functions and will produce an error message. Grid functions are updated in the order they are listed in the update declaration.

The second declaration will cause the compiler to produce the header for a routine called `evolver` which is expected to update grid functions `A`, `B`, and `C`. The body of the routine is left blank, to be filled in by the user. Following the reserved word `header`, comes a list of things to appear in the calling sequence for the function. A grid function name such as `A` above will cause all the time levels of `A` to be passed to the function. If `A` has three time levels (1, 0, -1), then they will be named `A_np1`, `A_n`, and `A_nm1` by default. The user can provide his own names to override the defaults as in the case of `B`. If only one name is provided (as for `C`), the time levels will be passed in as the elements of a single vector, the first component of which will be the *advanced* time level.

Other things that can appear in the header list are coordinates (such as `x`, `y`, and `z` above) and coordinate differentials (such as `dt`). Parameters can also be included in the list. Work arrays are declared like the final two parameters. First comes `auto` or `static`. Static work arrays are declared at the start of the program and persist throughout. Auto work arrays are allocated before the call to the update routine and are destroyed afterwards. Next comes the word `work` followed by the `#` symbol and an integer. This integer is tacked onto the end of the word work to form the name of the array. Finally comes an expression for the size of the work array enclosed in parentheses.

The thrid declaration is much like the second, except the body of the update routine is taken from the file named `myroutine.inc`.

8.1.2.5. The Loop Driver

There is one remaining kind of declaration—the *loop driver*. This declaration defines the overall method used for solving the equations. It consists of the reserved word `looper` followed by a name. The currently defined loop drivers are `standard` and `iterative`.

Declaring the `standard` loop driver means that the update routines will be called once for each time step. This driver can be used for a fully explicit system or when a user-written, external update routine is called.

Declaring the `iterative` loop driver means that the update routines will be called from a loop which first makes an initial guess at the advanced values and then continues to call the update routines until the norm of the residual is below a certain threshold. This driver is useful for implicit schemes.

Future loop drivers will may include a full Newton iteration, and will definitely include various adaptive options.

8.1.3. Expressions

Expressions are made up of objects separated by operators (see Table 8.1 for a list of operators in order of precedence). Objects include numbers, identifiers, functions, derivative operators, coordinates, and grid functions.

Numbers can be integer or floating point. Floating point numbers can be written with exponents as in `1.0e-3`. Identifiers are strings beginning with a letter or an underscore and containing letters, digits, or underscores. Identifiers may be names of grid functions, coordinates, or parameters.

A function is a function name followed by a parentheses-enclosed expression, such as `cos(3*r)`. The only currently recognized function names are: exp, log, tan, sin, cos, sinh, cosh, tanh, and sqrt.

A derivative operator is a name followed by an opening parenthesis, an expression, a coordinate list, and a closing parenthesis. It may also begin with the reserved word `expand` which tells RNPL to symbolically expand the derivative before making the operator substitution. For instance, `expand D_(a*b + c,r)` would expand to `D_(a,r)*b + a*D_(b,r) + D_(c,r)`.

A coordinate is a coordinate name followed by a spatial offset. For example,

```
r[1]
x[-5]
y{a*b+5}
```

are valid coordinates. The bracket-enclosed offsets are interpreted as in Section 8.1.2.1. That is, `r[1]` becomes r_{i+1}. The object inside the brackets must be an integer. The brace-enclosed expression is interpreted as an absolute index. That is, `y{a*b+5}` becomes y_{a*b+5}. The object in the braces is an arbitrary expression. If a coordinate name appears in an expression without a following offset, it will still be recognized as a coordinate and will be given an offset of zero.

A grid function is much like a coordinate except the name is preceded by a temporal offset in angle brackets and is followed by one spatial offset for each dimension. The following are examples of grid functions.

```
<0>A[0][-1]
<-1>B[1]{b+2}[0]
```

Also like a coordinate, a grid function name can appear without offsets.

There are three "types" of expressions possible in RNPL. A type 1 expression is completely arbitrary. A type 2 expression is a type 1 expression that contains no logical operators. A type 3 expression is a type 2 expression that contains no derivative operators, grid functions, or coordinates. Table 8.3 shows which expression types can be used in which areas.

8.2. Examples

Since languages are best learned by example, I will present two which illustrate most of RNPL's features.

8.2.1. 3D Wave Equation

Consider the linear wave equation in three dimensions. This is an initial-value, boundary-value problem which can be stated as follows:

$$\partial_t^2\phi = \partial_x^2\phi + \partial_y^2\phi + \partial_z^2\phi$$

$$\phi\left(x_{\min}, y, z, t\right) = 0$$

$$\phi\left(x_{\max}, y, z, t\right) = 0$$

$$\phi\left(x, y_{\min}, z, t\right) = 0$$

$$\phi\left(x, y_{\max}, z, t\right) = 0$$

$$\phi\left(x, y, z_{\min}, t\right) = 0$$

$$\phi\left(x, y, z_{\max}, t\right) = 0$$

$$\phi\left(x, y, z, 0\right) = A\exp\left(\left(x-c_x\right)^2/\delta_x^2\right)\exp\left(\left(y-c_y\right)^2/\delta_y^2\right)\exp\left(\left(z-c_z\right)^2/\delta_z^2\right)$$

Location	Type	Example
if statement	1	if (expr) then
residual	2	residual A { [1:1] := expr }
initialization	2	initialize A { [1:1] := expr }
absolute index	2	x{ expr }
index region	3	residual A { [1: expr]

Table 8.3. RNPL Expression Types

where $x_{\min} \le x \le x_{\max}$, $y_{\min} \le y \le y_{\max}$, and $z_{\min} \le z \le z_{\max}$.

Typically one would also specify $\dot{\phi}$, but RNPL doesn't allow this (see Section 9.4 for a full discussion of the RNPL initial data problem). To set this problem up with RNPL we must identify our requirements. We need one grid function, ϕ. We need difference operators for ∂_t^2, ∂_x^2, ∂_y^2, and ∂_z^2. We also need parameters for the initial data, namely c_x, c_y, c_z, A, δ_x, δ_y, and δ_z as well as parameters for the domain boundaries, $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$, $z_{\min}$, and $z_{\max}$.

We begin this RNPL program by specifying the parameters. The declarations look like this:

```
parameter float xmin := 0
parameter float xmax := 100
parameter float ymin := 0
parameter float ymax := 100
parameter float zmin := 0
parameter float zmax := 100
parameter float A    := 1.0
parameter float c_x  := 50.0
parameter float c_y  := 50
parameter float c_z  := 50
parameter float delta_x
parameter float delta_y
```

```
parameter float delta_z
```

Default values are optional.

Next, we define the coordinate system. The declaration looks like this:

```
rect coordinates t,x,y,z
```

The name `rect` is arbitrary, but should be descriptive.

Since we need a grid function, we must define a grid.

```
uniform rect grid g1 [1:Nx][1:Ny][1:Nz]
{xmin:xmax}{ymin:ymax}{zmin:zmax}
```

As stated in Section 8.1.1.3, the index and spatial ranges will be automatically defined if we leave them out.

We define our grid function to have three time levels so we can use the standard leap-frog operators to solve the equation. The definition is:

```
float phi on g1 at -1,0,1
```

Now come the operator definitions. We need four second derivatives, one for each coordinate.

```
operator D_LF(f,t,t) := (<1>f[0][0][0] - 2*<0>f[0][0][0] +
                        <-1>f[0][0][0])/(dt*dt)
operator D_LF(f,x,x) := (<0>f[1][0][0] - 2*<0>f[0][0][0] +
                         <0>f[-1][0][0])/(dx*dx)
operator D_LF(f,y,y) := (<0>f[0][1][0] - 2*<0>f[0][0][0] +
                         <0>f[0][-1][0])/(dy*dy)
operator D_LF(f,z,z) := (<0>f[0][0][1] - 2*<0>f[0][0][0] +
                         <0>f[0][0][-1])/(dz*dz)
```

Since we wish RNPL to produce the complete program, we must specify the partial differential equations. This is done by defining the residual.

```
evaluate residual phi {
          [1:Nx][1:Ny][1:1] := <1>phi[0][0][0] = 0;
        [1:Nx][1:Ny][Nz:Nz] := <1>phi[0][0][0] = 0;
          [1:Nx][1:1][1:Nz] := <1>phi[0][0][0] = 0;
        [1:Nx][Ny:Ny][1:Nz] := <1>phi[0][0][0] = 0;
          [1:1][1:Ny][1:Nz] := <1>phi[0][0][0] = 0;
        [Nx:Nx][1:Ny][1:Nz] := <1>phi[0][0][0] = 0;
  [2:Nx-1][2:Ny-1][2:Nz-1] := D_LF(phi,t,t) = D_LF(phi,x,x) +
                              D_LF(phi,y,y) + D_LF(phi,z,z)
                       }
```

The boundary conditions could also have been stated with a time derivative of ϕ, but this would have required another operator definition. The above method is the simplest.

To get RNPL to generate the initial data, we must provide an initialization for ϕ.

```
initialize phi {
  [1:Nx][1:Ny][1:Nz] := A*exp(-(x-c_x)^2/delta_x^2)*
                          exp(-(y-c_y)^2/delta_y^2)*
                          exp(-(z-c_z)^2/delta_z^2)
               }
```

We now instruct RNPL to solve the equation iteratively and to automatically generate the update routine.

```
looper iterative
auto update phi
```

8.2.2. "Shifted" Wave Equation

As a slightly more complicated example, let's consider the "shifted" wave equation in one dimension with periodic boundary conditions. We'll take the shift β to be a constant and the initial field configuration ϕ to be a left-moving Gaussian pulse. This problem can be stated as follows:

$$\partial_t^2\phi(x,t) = \left(1-\beta^2\right)\partial_x^2\phi(x,t) - 2\beta\,\partial_t\partial_x\phi(x,t)$$

$$\phi(x_{\min},t) = \phi(x_{\max},t)$$

$$\phi(x,0) = A\exp\left(-(x-c)^2/\Delta^2\right)$$

$$\partial_t\phi(x,0) = \frac{-2(x-c)}{\Delta^2}A\exp\left(-(x-c)^2/\Delta^2\right)$$

$$\beta(x) = 0.5$$

where $x_{\min} \leq x \leq x_{\max}$.

We can rewrite this equation in first order form by introducing the two auxiliary variables Φ and Π defined by:

$$\Phi \equiv \partial_x\phi\,,$$

$$\Pi \equiv \partial_t\phi - \beta\partial_x\phi\,.$$

In terms of these variables, the problem becomes:

$$\partial_t\Phi(x,t) = \partial_x(\beta\,\Phi + \Pi)$$

$$\partial_t\Pi(x,t) = \partial_x(\beta\,\Pi + \Phi)$$

$$\Phi(x,0) = \frac{-2(x-c)}{\Delta^2}A\exp\left(-(x-c)^2/\Delta^2\right)$$

$$\Pi(x,0) = \frac{-(x-c)}{\Delta^2}A\exp\left(-(x-c)^2/\Delta^2\right)$$

$$\Phi(x_{\min}, t) = \Phi(x_{\max}, t)$$

$$\Pi(x_{\min}, t) = \Phi(x_{\max}, t)$$

The RNPL program to solve this problem is shown below. It uses a two-level Crank-Nicholson difference scheme with numerical dissipation.

```
# This program solves 1D 1st order shifted wave equation

# with constant shift and periodic boundary conditions


parameter float xmin := 0

parameter float xmax

parameter float epsdis

parameter float c

parameter float A

parameter float delta


rec coordinates t,x

uniform rec grid g1 [0:Nx-1] {xmin:xmax}


float Phi on g1 at 0,1

float Pi on g1 at 0,1

float beta on g1

float phi on g1 at 0,1


operator D_PER(f,x) := (<1>f[1] - <1>f{Nx-2} + <0>f[1] - <0>f{Nx-2})/(4*dx)

operator D_CN(f,t) := (<1>f[0] - <0>f[0])/dt

operator D_CN(f,x) := (<1>f[1] - <1>f[-1] + <0>f[1] - <0>f[-1])/(4*dx)

operator D_CND(f,t) := (<1>f[0] - <0>f[0] +

  epsdis/16*(6*<0>f[0] + <0>f[-2] + <0>f[2] -4*(<0>f[-1] + <0>f[1])))/dt
```

```
operator D_CNDP1(f,t) := (<1>f[0] - <0>f[0] +
  epsdis/16*(6*<0>f[0] + <0>f{Nx-3} + <0>f[2] -4*(<0>f{Nx-2} + <0>f[1])))/dt
operator D_CNDP2(f,t) := (<1>f[0] - <0>f[0] +
  epsdis/16*(6*<0>f[0] + <0>f{Nx-2} + <0>f[2] -4*(<0>f[-1] + <0>f[1])))/dt
operator D_CNDP3(f,t) := (<1>f[0] - <0>f[0] +
  epsdis/16*(6*<0>f[0] + <0>f[-2] + <0>f{1} -4*(<0>f[-1] + <0>f[1])))/dt
operator AVG(f,t) := (<1>f[0] + <0>f[0])/2

evaluate residual Phi { [0:0] := D_CNDP1(Phi,t) = D_PER(beta*Phi + Pi,x);
                        [1:1] := D_CNDP2(Phi,t) = D_CN(beta*Phi + Pi,x);
                     [2:Nx-3] := D_CND(Phi,t) = D_CN(beta*Phi + Pi,x);
                  [Nx-2:Nx-2] := D_CNDP3(Phi,t) = D_CN(beta*Phi + Pi,x);
                  [Nx-1:Nx-1] := <1>Phi[0] = <1>Phi{0} }

residual Pi { [0:0] := D_CNDP1(Pi,t) = D_PER(beta*Pi + Phi,x);
              [1:1] := D_CNDP2(Pi,t) = D_CN(beta*Pi + Phi,x);
           [2:Nx-3] := D_CND(Pi,t) = D_CN(beta*Pi + Phi,x);
        [Nx-2:Nx-2] := D_CNDP3(Pi,t) = D_CN(beta*Pi + Phi,x);
        [Nx-1:Nx-1] := <1>Pi[0] = <1>Pi{0} }

residual phi { [0:Nx-1] := D_CN(phi,t) = AVG(Pi + beta*Phi,t) }

initialize beta { [0:Nx-1]:= .5 }
initialize Phi { [0:Nx-1]:= -2*(x-c)/delta^2*A*exp(-(x-c)^2/delta^2) }
initialize Pi { [0:Nx-1]:= -(x-c)/delta^2*A*exp(-(x-c)^2/delta^2) }
initialize phi { [0:Nx-1]:= A*exp(-(x-c)^2/delta^2) }

looper iterative
```

```
auto update Phi, Pi, phi
```

Notice that the periodic boundary conditions are enforced in the operators and the residuals by using an absolute index. Without this construct, such a boundary condition is impossible to implement except for a fixed size grid.

Chapter 9. RNPL: The Compiler

I have written a compiler for RNPL. This compiler is a complete implementation of the language as defined in Chapter 8. As new constructs and features are added to the language, the compiler will be updated to include support for them.

Other authors [23] have argued that compilers for numerical languages are best written in a language such as Maple or Mathematica because of their in-built symbolic manipulation capability. It is my opinion that compilers (including ones for math-oriented languages such as RNPL) should be written in a traditional programming language such as C in combination with compiler writing tools such as lex and yacc (readers interested in using lex and yacc should consult [16]).

I first planned to implement the RNPL compiler in Maple. However, it soon became clear that such a language is not suited to the task. On the one hand, RNPL does require some symbolic manipulation (simple algebra and differentiation), a task which Maple can perform well. On the other hand, compilers require complex data structures and excellent text handling, both tasks which are better suited to a general purpose programming language. I decided to write the compiler in C, using lex and yacc for the parser. I wrote my own symbolic manipulation routines. As it turned out, the time spent implementing the symbolic capabilities was **much** less than the time spent on code generation, thus justifying my decision.

It can certainly be argued that C++ would be better suited to the task than C, however I decided to use C due to its wider availability on the target platforms. A future version of the compiler may use C++.

9.1. Compiler Assumptions

As discussed in Chapter 8, RNPL has no “executable” statements. Because of this, it is up to the compiler to decide how to translate the source into an executable program (or portion thereof). In this way, the user is shielded as much as possible from the details of programming and can concentrate on the equations. In order to successfully generate a program, the compiler must make certain assumptions which place some constraints on the form of the source. Although some of these assumptions seem to violate the strong typing rules of the language, in actuality, they simply move the responsibility for some declarations from the user to the compiler. That is, all data objects are still declared, just some are declared automatically. If a name is multiply defined, the first definition will be kept and subsequent ones will be discarded. If the objects with the same name are of the same type, for instance two parameters named X, then no error will be reported. However, if X is first declared to be a parameter and later redeclared to be a grid function, the compiler will report an error.

9.1.1. Coordinates and Differentials

Coordinate differentials are automatically defined by the compiler. They are given the same names as the coordinates except they begin with a lower case d. Thus, if t and x are coordinates, dt and dx will be coordinate differentials. At run time the coordinate differentials are assigned values based on the grids. For instance, if an RNPL program contains the following definitions:

```
rectangular coordinates t,x,y


uniform rectangular grid G [1:Nx][0:Ny-1]
                       {xmin:xmax}{ylow:yhigh}
```

then the spatial coordinate differentials will be assigned values using

$$dx = \frac{xmax - xmin}{Nx - 1}$$

and

$$dy = \frac{yhigh - ylow}{Ny - 1},$$

while the time coordinate differential will be assigned by

$$dt = \lambda\sqrt{\frac{dx^2 + dy^2}{2}},$$

where λ is the Courant factor (see below).

When there is more than one grid defined, the compiler uses the first grid which uses a given coordinate to define that coordinate's differential. If along with the above definitions, an RNPL program contains

```
uniform rectangular[x] grid g [1:N] {min:max}
```

then dx will still be defined as above. The user must construct his own grid spacing variable for grid g. On the other hand, if the definition for g came before the definition for G, dx would be assigned from g by $dx = (\min - \max)/(N - 1)$, while dy would still be defined from G, since that is the first grid which uses y.

Along with the coordinate differentials, the grid sizes are defined automatically. These are named like the differentials except with an initial upper case N instead of the d. Also like the differentials, the grid sizes are not assigned values by the user. Their values are calculated from the *grid base* parameters (see below). Although the user can use any expression he likes to define the index regions, using the grid sizes is best since it allows for easy convergence testing by changing only the *level* parameter (see below).

9.1.2. "Special" Parameters

There are several parameters which are needed by every RNPL program. Instead of forcing the user to always define these, the compiler takes care of them. These parameters are listed in Table 9.1. Most of their uses are obvious. The others will be explained here or in the following sections.

The *level* parameter and the grid bases can be used together for easy convergence testing. Assume we have defined the coordinates t, x, y, and z. Then at run time, the grid sizes will be set as follows:

Name	Type	Default	Use
start_t	float	0	start time
s_step	int	0	starting iteration
iter	int	100	number of iterations
epsiter	float	1.0e-5	iteration threshold
fout	int	0	file output (0 no, 1 yes)
ser	int	0	fs output (0 no, 1 yes)
lambda	float	.5	Courant factor
output	ivec	*-*/1	output control
in_file	string	none	name of file from which initial data will be read
out_file	string	none	name of file to which final state will be written
level	int	0	refinement level
tag	string	""	prepend symbol for grid function names
N*c*0	int	0	base number of grid points for the coordinate named *c* (there is one for each spatial coordinate)

Table 9.1. Special Parameters

$$Nx = Nx0 \cdot 2^{level} + 1$$

$$Ny = Ny0 \cdot 2^{level} + 1$$

$$Nz = Nz0 \cdot 2^{level} + 1.$$

That is, if *Nx0* is 100 and *level* is 0, then *Nx* will be set to 101. If *level* is 1, then *Nx* will be set to 201. If any grid base is an odd number, then the above procedure will not give proper lengths for a convergence series. In this case, the grid size will be set to the grid base and a warning message will be printed. This way, a user can do a single run with an arbitrary grid size.

If a grid has been declared without using the grid sizes, then its size will remain unaffected by changes to *level*.

9.2. Language Modifications

Feedback from use by myself and others has resulted in several modifications of the initial language design which make it more effective. These modifications are discussed below.

One of the initial design goals for RNPL was to allow users to easily convert existing programs to RNPL so they could take advantage of the built-in features. Initially, this was going to be done by having the user directly edit the RNPL-generated code. This proved too burdensome, so the update declaration was modified to allow automatic header generation and code inclusion.

In order to have the compiler output FORTRAN 77, I had to add *constant* parameters and *system* parameters. It was also necessary to either make RNPL completely case-insensitive, or case-insensitive when producing FORTRAN output. I chose the latter option.

Constant parameters are needed because of FORTRAN's lack of support for globals. They behave exactly like regular parameters except they can not be passed to update routines or residual evaluators in the argument list. They are placed in a separate common block from the other parameters and globals and are declared in every function.

System parameters are needed because of FORTRAN's lack of support for dynamically allocated memory. There is currently only one system parameter defined—`memsiz`. This parameter has the default value of 2000000. It is the size of the FORTRAN program's heap in doubles. It can be changed with a declaration like:

```
system parameter int memsiz := 8000000
```

A few syntactic changes were made to RNPL at user's requests. These were support for all capital reserved words, the # in the work array declaration, and the optional semicolon to end a residual block.

9.3. Equation Solver

The primary job of an RNPL program is to solve a system of evolution equations. The equations are specified through the residuals and it is up to the compiler to decide how to solve them. The solution is generated by symbolically solving each grid function's residual for its advanced value.

This method is exactly what is required for solving explicit schemes. Using the iterative loop driver, one can use this method to solve implicit schemes as well. Such an iterative method may in fact be sufficient for all implicit schemes, though some early attempts to use this method on the fully-averaged implicit scheme failed to converge.

In the future, I will add support for a global Newton iteration which should solve any implicit scheme. Although such a scheme is no more difficult to support than the current scheme (from the compiler's point of view), it requires a banded matrix solver. Due to the large number of architectures that RNPL supports (these include serial, vector, and massively parallel machines), I decided to wait on the matrix routines until it was clear that they were needed.

9.4. Initial Data Generation

The weakest part of the compiler is its initial data support. It only allows analytic initial data to be specified. However, the initial data problem is very different from the evolution problem that RNPL was designed to solve. If not known analytically, initial data is usually solved for from elliptic equations. Techniques to solve elliptic equations are very different from techniques used to solve hyperbolic equations.

There are also problems with initial data that are inherent to finite differencing. For instance, assume we are attempting to solve the one dimensional linear wave equation in flat space. This equation is $\partial_t^2\phi = \partial_x^2\phi$. The initial state of the system is specified by giving $\phi(x,0)$ and $\dot{\phi}(x,0)$.

In RNPL, we can only give $\phi(x,0)$. If we wish to use a three-level scheme to solve this equation, then we need two time levels of initial data. RNPL will find the second time level through an iterative procedure using the evolution equations. This will result in time-symmetric initial data which contains both an ingoing and an outgoing piece. If this is not what we want, we must modify the compiler-generated initial data generator to produce our own data.

However, assume we wish to use a two-level scheme. In this case we can still only give $\phi(x,0)$. However, that is all the data we can possibly give. This is a problem with using the two-level scheme. It does not capture both degrees of freedom available in the physical problem.

Now assume we decide to rewrite the problem in first order form. The equations of motion are $\partial_t \Phi = \partial_x \Pi$ and $\partial_t \Pi = \partial_x \Phi$, with $\Phi \equiv \partial_x \phi$ and $\Pi \equiv \partial_t \phi$. Now using a two level scheme we can specify one time level for each function. This allows us to completely specify the initial configuration.

A three-level scheme for this problem will require four levels of initial data, two for each function. RNPL will only allow us to specify two levels and will solve for the other two using an iterative procedure. In this case, unlike the second order problem above, this is the correct way of generating the extra time levels since the physical problem only contains two degrees of freedom.

These examples show that RNPL will correctly handle the specification of analytic initial data for systems written in first order form and using two or three-level finite difference schemes for solution. A simple modification to RNPL would allow the user to specify multiple time levels of initial data for each grid function. This would allow RNPL to correctly handle initial data for systems written in higher order form.

9.5. Parameter Files

At run time, RNPL generated programs look for a parameter file. This is an ASCII file consisting of arbitrary text. The RNPL program will scan this file for lines of the form `name := value`, where `name` is the name of a parameter, and `value` is the value of the parameter given in the same form as default values in RNPL source. If the program finds such a line for a parameter, it will set that parameter to the value in the file. If not, it will use the default value for the parameter. If the parameter has no default, the program will exit with an error message.

Here is an example of a parameter file:

```
This is a parameter file
tag := "a_"
level := 1
Nx0 := 100
xmin := -10
This is a comment
in_file := "init_data.hdf"
out_file := "dump.hdf"
```

9.6. Output Control

RNPL generated programs allow the user to dynamically control output during execution. At any time, the user can interrupt the program and turn output on or off for any grid function, as well as control the frequency of output.

By default, output is controlled by three parameters and an attribute. The parameter *fout* can be set to zero or one. If it is set to one, then output is sent to HDF files. If the parameter *ser* is set to one, then output is sent to Matthew Choptuik's *vs* graphics server. Since *vs* only handles one dimensional functions, only one dimensional grid functions will be sent from the program.

The parameter *output* is an index vector. It controls when the program generates output. For example, if the interesting dynamics occur after iteration 100, output could be set like:

```
output := *-100/50,101-150/2,151-*/50
```

This would output only once every 50 time steps during the first part of the calculation, every other time step during the dynamic part, and then every 50 time steps till the end. The initial * is set to `s_step` and the final * is set to `iter`.

There is one automatically defined attribute. It controls which grid functions are output. It is called `out_gf` and has encoding `encodeone`. Any of its elements can be set to one to enable output for that grid function or to zero to disable output.

9.7. Check Pointing

Check pointing provides a way for calculations to be stopped during execution and then restarted at a later time without losing any information. All RNPL generated programs automatically dump state upon completion. If a calculation needs to be stopped during execution, it can be interrupted and told to dump state. The state is saved in the file whose name is given by the parameter `out_file`. The state file is identical in format to an initial data file. Thus, to restart, the user simply has to set the parameter `in_file` to the name of the state file and rerun the program. The parameters `start_t` and `s_step` are read from the state file, so their values in the parameter file do not need to be changed.

References

[1] Annninos, P. et. al. Horizon boundary condition for black hole spacetimes. *Phys. Rev. D* **51**, 5562–78 (1995).

[2] Bardeen, J. M. and T. Piran. General Relativistic Axisymmetric Rotating Systems: Coordinates and Equations. *Phys. Rep.* **96**, 205–250 (1983).

[3] Bechmann, O. and O. Lechtenfeld. Exact black-hole solution with self-interacting scalar field. *Class. Quant. Grav.* **12**, 1473–81 (1995).

[4] Bekenstein, J. D. Nonexistence of Baryon Number for Static Black Holes. *Phys. Rev. D* **5**, 1239–46 (1972).

[5] Choptuik, M. W. and W. G. Unruh. No title. Unpublished Work (1988).

[6] Choptuik, M. W. *A Study of Numerical Techniques for Radiative Problems in General Relativity*. Ph.D. Dissertation, The University of British Columbia, 1986.

[7] Choptuik, M. W. No title. Unpublished Notes (1991).

[8] Choptuik, M. W. Consistency of finite-difference solutions of Einstein's equations. *Phys. Rev. D* **44**, 3124–35 (1991).

[9] De Felice, F. and C. J. S. Clarke. *Relativity on curved manifolds*. Cambridge University Press, Cambridge, 1990.

[10] DeWitt, B. S. and R.W. Brehme. Radiation Damping in a Gravitational Field. *Ann. Phys. (N.Y.)* **9**, 220–59 (1960).

[11] Gundlach, C. et. al. Late-time behavior of stellar collapse and explosions. I. Linearized perturbations. *Phys. Rev. D* **49**, 883–9 (1994).

[12] Gundlach, C. et. al. Late-time behavior of stellar collapse and explosions. II. Nonlinear evolution. *Phys. Rev. D* **49**, 890–9 (1994).

[13] Kurki-Suonio, H. et. al. Inhomogeneous inflation: Numerical evolution. *Phys. Rev. D* **48**, 3611–24 (1993).

[14] Kingston, J. H. A User's Guide to the Lout Document Formatting System. Tech. Rep. (1995), Basser Department of Computer Science, The University of Sydney, Sydney, Australia.

[15] Liu, H. and B. Mashhoon. On the spectrum of oscillations of a Schwarzschild black hole. *Preprint* (1995).

[16] Levine, J. R. et. al. *lex & yacc*. O'Reilly & Associates, Sebastopol, CA, 1990.

[17] Matzner, R. A. Private discussion. No publisher (1995).

[18] Misner, C. W. et. al. *Gravitation.* Freeman, New York, 1973.

[19] Price, R. H. Nonspherical Perturbations of Relativistic Gravitational Collapse. I. Scalar and Gravitational Perturbations. *Phys. Rev. D* **5**, 2419–38 (1972).

[20] Regge, T. and J. A. Wheeler. Stability of a Schwarzschild Singularity. *Phys. Rev.* **108**, 1063–9 (1957).

[21] Richardson, L. F. The Approximate Arithmetical Solution by Finite Differences of Physical Problems Involving Differential Equations with an Application to the Stresses in a Masonry Dam. *Phil. Trans. Roy. Soc.* **210**, 307–57 (1910).

[22] Seidel, E. and W. Suen. Towards a Singularity-Proof Scheme in Numerical Relativity. *Phys. Rev. Let.* **69**, 1845–48 (1992).

[23] Thornburg, J. *Numerical Relativity in Black Hole Spacetimes*. Ph.D. Dissertation, The University of British Columbia, 1993.

[24] Wald, R. M. *General Relativity*. University of Chicago Press, Chicago, 1984.

VITA

Robert Lee Marsa was born to Lynette Maurine Marsa and Gordon Lee Marsa on September 19, 1969 in Pontiac Michigan. After completing high school at Fletcher Academy, Fletcher, North Carolina, in 1987, Robert attended Southern College in Collegedale, Tennessee. He graduated in May of 1991 with a Bachelor of Science in Mathematics and Physics and an Associate of Science in Engineering. On May 26, 1991 he was married to Meri Anissa Housley. In August, he entered the Graduate School of The University of Texas. He worked at the university's Applied Research Laboratories until January of 1995 when he became a Research Assistant at the Center for Relativity. He has accepted a position as Research Associate in the Department of Physics and Astronomy at the University of Pittsburgh. This position begins in January 1996.

Permanent address: 5916 Douglas St. Pittsburgh, PA 15217

This dissertation was typeset by the author using Basser Lout [14].