Massive Electromagnetically Coupled Complex Scalar Fields

Roman Petryk

December 16, 2000

Abstract

We solve the dynamics of the massive complex scalar field coupled to electromagnetism. The observed dynamics show marked differences from those of the uncoupled Klein-Gordon system. The numerical method displays the expected energy conservation, and the fields each exhibit convergence factors near or greater than 4.

1 Introduction

In this article, we will see how to solve for the evolution of the massive complex scalar field coupled to electromagnetism. We will do so numerically using finite difference techniques, and employing the Rapid Numerical Prototyping Language (RNPL) developed by Marsa and Choptuik ([1] is a good reference).

Although this system has been previously studied by Hod and Piran [2], and Liebling [3], our investigation will differ in a few fundamental ways. While [2] involved a gravitationally coupled *massless* scalar field, our study will involve a *massive* field evolving on a flat spacetime background. And while [3] studies a massive scalar field in axisymmetry, we will investigate the system in spherical symmetry (and with a somewhat simpler potential). Both we and [3] use a convention for the electromagnetic stress-energy tensor that differs from those of [2] and [4] by a constant factor. As well, our convention for the sign of charge is opposite that of [2] and [4].

An informative introduction to the electromagnetically coupled complex scalar system in flat spacetime can be found in [5]. However, the reader should be warned that the exposition contained therein is not written in covariant form. Furthermore, a different metric signature is in effect. Specifically, we will be using the signature of [4], which is $(-,+,+,+)$. Furthermore, our Greek tensor indices will range 0,1,2,3, and Latin indices will range 1,2,3.

In *Section 2* we will derive the equations of motion for our system and cast them in first order form for flat spherically symmetric spacetime. We will then discuss the initial constraints on our system and its boundary conditions. *Section 3* will discuss the numerical implementation of the derived equations, and tests on our finite difference code. The *Appendices* contain listings of the code itself. Finally, MPEG format video clips showing select field evolutions are available for download off the World Wide Web at http://laplace.physics.ubc.ca/Students/petryk/index.html.

2 Lagrangian Formulation

In flat spacetime, the *real* scalar field Lagrangian

$$\mathcal{L} = -\frac{1}{2}(\nabla_\mu \phi)(\nabla^\mu \phi) - \frac{1}{2}m^2\phi^2 \tag{1}$$

along with the Euler-Lagrange equations

$$\frac{\partial \mathcal{L}}{\partial \phi} - \nabla_\mu \left[\frac{\partial \mathcal{L}}{\partial(\nabla_\mu \phi)}\right] = 0 \tag{2}$$

give rise to the Klein-Gordon equation

$$(-\nabla^\mu \nabla_\mu + m^2)\phi = 0. \tag{3}$$

However, the *complex* field Lagrangian

$$\mathcal{L} = -(\nabla_\mu \phi - ieA_\mu \phi)(\nabla^\mu \phi^* + ieA^\mu \phi^*) - m^2\phi\phi^* - \frac{1}{4}F^{\mu\nu}F_{\mu\nu} \tag{4}$$

along with the Lorentz Gauge condition

$$\nabla_\mu A^\mu = 0 \tag{5}$$

and the set of Euler-Lagrange equations

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial A_\nu} - \nabla_\mu \left[\frac{\partial \mathcal{L}}{\partial(\nabla_\mu A_\nu)}\right] &= 0, & (6)\\
\frac{\partial \mathcal{L}}{\partial \phi} - \nabla_\mu \left[\frac{\partial \mathcal{L}}{\partial(\nabla_\mu \phi)}\right] &= 0, & (7)\\
\frac{\partial \mathcal{L}}{\partial \phi^*} - \nabla_\mu \left[\frac{\partial \mathcal{L}}{\partial(\nabla_\mu \phi^*)}\right] &= 0, & (8)
\end{aligned}$$

yield the following set of equations of motion

$$\begin{aligned}
\nabla^\mu \nabla_\mu A_\nu &= +ie(\phi^* \nabla_\nu \phi - \phi \nabla_\nu \phi^*) + 2e^2 A_\nu \phi\phi^*, & (9)\\
\nabla^\mu \nabla_\mu \phi &= +2ie(\nabla_\mu \phi)A^\mu + e^2 A_\mu A^\mu \phi + m^2\phi, & (10)\\
\nabla^\mu \nabla_\mu \phi^* &= -2ie(\nabla_\mu \phi^*)A^\mu + e^2 A_\mu A^\mu \phi^* + m^2\phi^*, & (11)
\end{aligned}$$

where $\mu, \nu = 0, 1, 2, 3$, A_μ is the the electromagnetic four-potential, $F_{\mu\nu}$ is the electromagnetic field-strength tensor defined according to

$$F_{\mu\nu} = \nabla_\mu A_\nu - \nabla_\nu A_\mu, \tag{12}$$

and ϕ and ϕ^* are respectively the complex scalar field and its conjugate. By construction, the complex scalar field Lagrangian is invariant under gauge transformations of the second kind [5].

While the derivation of (9), (10) and (11) are trivial, it would be prudent to make mention of an important fact related to the equation of motion for A_μ. Differentiating the electromagnetic field-strength tensor (12) and using Lorentz gauge condition (5), we obtain

$$\begin{aligned} \nabla^\mu F_{\mu\nu} &= \nabla^\mu(\nabla_\mu A_\nu - \nabla_\nu A_\mu) \\ &= \nabla^\mu \nabla_\mu A_\nu. \end{aligned} \tag{13}$$

Therefore, (9) into (13) yields,

$$\nabla_\mu F^{\mu\nu} = \mathcal{J}^\nu \tag{14}$$

where

$$\mathcal{J}^\nu = ie(\phi^* \nabla^\nu \phi - \phi \nabla^\nu \phi^*) + 2e^2 A^\nu \phi\phi^*. \tag{15}$$

Now, since the antisymmetry of $F_{\mu\nu}$ requires

$$\nabla_\mu \nabla_\nu F^{\mu\nu} = \nabla_\nu \mathcal{J}^\nu = 0, \tag{16}$$

we interpret $\mathcal{J}^\nu$ to be the conserved *covariant* current. e is understood to be the coupling constant between complex scalar and electromagnetic fields.

Returning to the equations of motion themselves, we will find it convenient to re-express the complex scalar field in terms of its real and imaginary components. Thus, introducing two *real* fields ϕ_1 and ϕ_2 (i.e., $\Im(\phi_1) =$

$\Im(\phi_2) = 0)$ we write

$$\phi = \phi_1 + i\phi_2, \tag{17}$$

$$\phi^* = \phi_1 - i\phi_2. \tag{18}$$

In terms of this representation, (9, 10, 11) become

$$\nabla^\mu \nabla_\mu A_\nu = -2e(\phi_1 \nabla_\nu \phi_2 - \phi_2 \nabla_\nu \phi_1) + 2e^2 A_\nu ({\phi_1}^2 + {\phi_2}^2), \tag{19}$$

$$\nabla^\mu \nabla_\mu \phi_1 = -2e(\nabla_\mu \phi_2) A^\mu + e^2 A_\mu A^\mu \phi_1 + m^2 \phi_1, \tag{20}$$

$$\nabla^\mu \nabla_\mu \phi_2 = +2e(\nabla_\mu \phi_1) A^\mu + e^2 A_\mu A^\mu \phi_2 + m^2 \phi_2. \tag{21}$$

2.1 Flat Spherical Symmetry

Thus far, our formulation has been completely general. We now focus on flat, spherically symmetric spacetime with metric signature $(-,+,+,+)$.

Since the flat spacetime metric in spherical polar coordinates (t, r, θ, ϕ) is

$$ds^2 = -dt^2 + dr^2 + r^2 d\theta^2 + r^2 sin^2\theta d\phi^2, \tag{22}$$

the nonzero components of the connection are

$$\Gamma^r_{\theta\theta} = -r, \tag{23}$$

$$\Gamma^r_{\phi\phi} = -r sin^2(\theta), \tag{24}$$

$$\Gamma^\theta_{r\theta} = \Gamma^\theta_{\theta r} = \frac{1}{r}, \tag{25}$$

$$\Gamma^\theta_{\phi\phi} = -sin(\theta)cos(\theta), \tag{26}$$

$$\Gamma^\phi_{r\phi} = \Gamma^\phi_{\phi r} = \frac{1}{r}, \tag{27}$$

$$\Gamma^\phi_{\theta\phi} = \Gamma^\phi_{\phi\theta} = \frac{cos(\theta)}{sin(\theta)}. \tag{28}$$

Furthermore, spherical symmetry requires rotations in the θ- and ϕ-coordinate directions to be isometries of the spacetime. Equations (19, 20, 21) and Lorentz gauge condition (5) thereby reduce to

$$\begin{aligned}
\partial_t^2 A_t = &\quad \partial_r^2 A_t + 2r^{-1}\partial_r A_t + 2e(\phi_1\partial_t\phi_2 - \phi_2\partial_t\phi_1) \\
&\quad -2e^2 A_t({\phi_1}^2 + {\phi_2}^2), \qquad (29)\\
\partial_t^2 A_r = &\quad \partial_r^2 A_r + 2r^{-1}\partial_r A_r - 2r^{-2}A_r + 2e(\phi_1\partial_r\phi_2 - \phi_2\partial_r\phi_1) \\
&\quad -2e^2 A_r({\phi_1}^2 + {\phi_2}^2), \qquad (30)\\
\partial_t^2 \phi_1 = &\quad \partial_r^2 \phi_1 + 2r^{-1}\partial_r\phi_1 - m^2\phi_1 - 2e((\partial_t\phi_2)A_t - (\partial_r\phi_2)A_r) \\
&\quad -e^2(-A_t^2 + A_r^2)\phi_1, \qquad (31)\\
\partial_t^2 \phi_2 = &\quad \partial_r^2 \phi_2 + 2r^{-1}\partial_r\phi_2 - m^2\phi_2 - 2e(-(\partial_t\phi_1)A_t + (\partial_r\phi_1)A_r) \\
&\quad -e^2(-A_t^2 + A_r^2)\phi_2, \qquad (32)
\end{aligned}$$

and

$$\partial_t A_t = \partial_r A_r + 2r^{-1}A_r. \qquad (33)$$

To cast the set of *second* order differential equations (29, 30, 31, 32, 33) in *first* order form, we introduce auxiliary fields

$$\begin{aligned}
\Pi_{Ai} &= \partial_t A_i, &(34)\\
\Omega_{Ai} &= \partial_r A_i, &(35)\\
\Pi_{\phi j} &= \partial_t \phi_j, &(36)\\
\Omega_{\phi j} &= \partial_r \phi_j, &(37)
\end{aligned}$$

where $i = t, r$ and $j = 1, 2$. We thereby obtain

$$\begin{aligned}
\partial_t \Pi_{At} = &\quad \partial_r\Omega_{At} + 2r^{-1}\Omega_{At} + 2e(\phi_1\Pi_{\phi 2} - \phi_2\Pi_{\phi 1}) \\
&\quad -2e^2 A_t({\phi_1}^2 + {\phi_2}^2), \qquad (38)\\
\partial_t \Pi_{Ar} = &\quad \partial_r\Omega_{Ar} + 2r^{-1}\Omega_{Ar} - 2r^{-2}A_r + 2e(\phi_1\Omega_{\phi 2} - \phi_2\Omega_{\phi 1})
\end{aligned}$$

$$-2e^2 A_r({\phi_1}^2 + {\phi_2}^2), \qquad (39)$$

$$\begin{aligned}
\partial_t \Pi_{\phi 1} = \ & \partial_r \Omega_{\phi 1} + 2r^{-1}\Omega_{\phi 1} - m^2\phi_1 - 2e(+\Pi_{\phi 2} A_t - \Omega_{\phi 2} A_r) \\
& -e^2(-{A_t}^2 + {A_r}^2)\phi_1, \qquad (40) \\
\partial_t \Pi_{\phi 2} = \ & \partial_r \Omega_{\phi 2} + 2r^{-1}\Omega_{\phi 2} - m^2\phi_2 - 2e(-\Pi_{\phi 1} A_t + \Omega_{\phi 1} A_r) \\
& -e^2(-{A_t}^2 + {A_r}^2)\phi_2, \qquad (41)
\end{aligned}$$

and

$$\Pi_{At} = \Omega_{Ar} + 2r^{-1} A_r. \qquad (42)$$

2.2 The Complete Set of Equations

Now, while (38, 39, 40, 41) are the equations of motion for our system, they amount to a system of four equations in twelve unknowns. Clearly, we need eight more equations. Four of these eight equations are simply the commutation relations between the partial derivatives,

$$\begin{aligned}
\partial_r \Pi_{At} &= \partial_t \Omega_{At}, & (43) \\
\partial_r \Pi_{Ar} &= \partial_t \Omega_{Ar}, & (44) \\
\partial_r \Pi_{\phi 1} &= \partial_t \Omega_{\phi 1}, & (45) \\
\partial_r \Pi_{\phi 2} &= \partial_t \Omega_{\phi 2}. & (46)
\end{aligned}$$

The remaining set of four equations can be chosen in one of two ways. We can either define the fields to be the integrals of their space derivatives, or we can we can define them to be the integrals of their associated time derivatives. For simplicity, we will choose the latter. That is, we will employ *free evolution* via

$$\partial_t A_t = \Pi_{At}, \qquad (47)$$

$$\partial_t A_r = \Pi_{Ar}, \tag{48}$$

$$\partial_t \phi_1 = \Pi_{\phi 1}, \tag{49}$$

$$\partial_t \phi_2 = \Pi_{\phi 2}. \tag{50}$$

Equations (38) through (41) and (46) through (50) now constitute the complete set of twelve equations in twelve unknowns needed to solve for the evolution of our physical system.

2.3 Boundary Conditions

Specifying the full set of differential equations, in itself, is not enough. We still need to provide appropriate boundary conditions for our fields.

We start by noting that the spatial derivatives of the scalar fields must vanish at the origin of spherical symmetry if regularity is to be observed. Also, the spatial component of the vector must vanish at the origin of coordinates. The following heuristic argument shows why this must be true. Suppose that each of the fields can be expanded as a power series in radial coordinate r. Then, if the leading power of each of the fields is given by

$$cr^p, \tag{51}$$

where c is a constant and p is some number, substitution into (38, 39, 40, 41) shows that the equations of motion will be nonsingular at $r = 0$ only if $p = 0$ for A_t, ϕ_1 and ϕ_2, and if $p = 1$ for A_r.

Now, although our system of partial differential equations will be well-behaved away from the $r = 0$ coordinate origin, our computational domain will necessarily be finite. We are therefore forced to consider restrictions on the behaviour of our fields at some maximum value of coordinate r, which we label $r = r_{max}$. Many different conditions could be chosen, but for our purposes we will want to place *outgoing radiation conditions* at $r = r_{max}$. For a decoupled ($e = 0$) massless ($m = 0$) system, an exact (*Sommerfeld*) outgoing radiation condition can be applied. However, because of dispersion, we can only employ approximate outgoing radiation conditions via

$$\Pi_{At} = -\Omega_{At} - r^{-1}A_t, \tag{52}$$

$$\Pi_{Ar} = -\Omega_{Ar} - r^{-1}A_r, \tag{53}$$

$$\Pi_{\phi 1} = -\Omega_{\phi 1} - r^{-1}\phi_1, \tag{54}$$

$$\Pi_{\phi 2} = -\Omega_{\phi 2} - r^{-1}\phi_2, \tag{55}$$

As it turns out, these conditions are sufficient for the initial data we will be employing in our computations, in that they will not noticeably reflect any of the fields reaching $r = r_{max}$.

2.4 Initial Data, Gauss' Law, and Field Decomposition

Now that we have obtained a complete set of equations (38, 39, 40, 41, 42) and boundary equations to solve for the evolution of our system, we must solve the *initial value problem* for our system. That is, we must find an appropriate set of conditions on the *initial spacelike hypersurface* that allow us to determine the state of the system on some *final spacelike hypersurface*. Taken together, the initial data problem, and evolution of the system from initial state to final state, constitutes the *Cauchy problem* of our coupled complex scalar system.[1]

Consider some general metric with coordinates (x^0, x^1, x^2, x^3) on a flat 4-dimensional spacetime. In the Cauchy problem there exists much freedom in

[1] The discussion of this section will closely follow that of [9], to which point the reader for details.

choosing the initial, 3-dimensional, spacelike hypersurface Σ_0 for this spacetime. For convenience, we take it to be orthogonal to the timelike Killing vector field k^μ. Now, choosing the normalization $k^\mu k_\mu = -1$ and adapting our coordinates so that $x^0 = t$, the Killing vector field can be expressed $\frac{\partial}{\partial t}$. Furthermore, there exists an entire family of such timelike Killing vector orthogonal hypersurfaces—they are defined by the complete set of surfaces of constant t. We label this family of *maximally slicing*[2] hypersurfaces Σ_t.

In this maximally slicing formulation, the flat spacetime metric can be written

$$ds^2 = -dt^2 + f_{ij}dx^i dx^j, \tag{56}$$

where $i, j = 1, 2, 3$ and f_{ij} is the flat spatial hypermetric. Notice that metric (22) is already of this form. Its spatial component,

$$dr^2 + r^2 d\theta^2 + r^2 sin^2\theta d\phi^2, \tag{57}$$

thus defines the hypermetric on Σ_t for our flat spherically symmetric spacetime. The Cauchy problem then amounts to solving equations (38, 39, 40, 41, 42) on Σ_t given the data on Σ_0. We must now figure out how to determine the initial data, and what constraint equations, if any, it must obey.

In their article [9], York and Piran demonstrate how a decomposition of the electric field allows us to rewrite Gauss' law in the form of Poisson's equation, thereby yielding an easily soluble initial data constraint. In detail, on some hypersurface Σ_t, the electromagnetic four-potential components can be written as a scalar potential Φ and a three-vector potential $A_i = f_{ij}A^j$. That is,

$$A_\mu = (-\Phi, A_i). \tag{58}$$

[2] Although it is beyond our present scope to investigate details in curved spacetime, it is worth mentioning that maximal slicing is often a convenient choice for generally relativistic Cauchy problems.

Similarly, the conserved covariant current can be decomposed

$$\mathcal{J}_\mu = (-\rho, j_i), \tag{59}$$

where charge density ρ is the hyperplane-orthogonal part of $\mathcal{J}_\mu$, and current density $j_i = f_{ij} j^j$ is the hyperplane-tangential component of $\mathcal{J}_\mu$. Having decomposed the four-current, we can write Gauss' Law for the electric field as[3]

$$\nabla^i E_i = \rho, \tag{60}$$

where ∇^i is the covariant derivative for spatial metric f_{ij}, and $E_i = f_{ij} E^j$ is the electric field. The initial value problem for electromagnetism then amounts to specifying the fields on Σ_0 subject to constraint equation (60).

Having stated the initial value problem, York and Piran [9] go on to show how we can solve constraint equation (60) by decomposing the electric field into longitudinal (*divergence full*) E_i^L and transverse (*divergence free*) E_i^T parts,

$$E_i = E_i^L + E_i^T. \tag{61}$$

First, field E_i is defined in terms of the gradient of a scalar function U. Specifically,

$$E^i = -\nabla^i U. \tag{62}$$

Then, equating (62) and (61), and taking the divergence of the resulting equation, we obtain

$$\nabla^i E_i^L \equiv -\nabla^i \nabla_i U, \tag{63}$$

$$\nabla^i E_i^T \equiv 0. \tag{64}$$

Then, substitution into (60) yields

$$\nabla^i \nabla_i U = -\rho, \tag{65}$$

[3]The usual factor of 4π is absent because of the convention we use in the Lagrangian.

which is a second order elliptic equation—the only constraint (outside of gauge conditions) on the initial conditions of the physical system.

The electromagnetic initial value problem has thus been reduced to a single equation (65). Now, while the longitudinal component E_i^L is explicitly involved in the solution of U, the transverse component E_i^T (which represents the radiative degrees of freedom) is freely specifiable. We could just as well choose it to be vanishing (as we should for spherically symmetric systems). Then, given charge density ρ, we can solve for U via (65). Having found U, we can then find E_i via (62). Meanwhile, we have the freedom to specify any reasonable initial values for A_i.[4]

We now consider initial data constraint (65) for the flat, spherically symmetric, complex scalar field problem. First, we calculate the conserved charge density ρ. Using (15) followed by (17, 18), and then (36), we explicitly obtain

$$\begin{aligned}\mathcal{J}^t &= ie(\phi^*\nabla^t\phi - \phi\nabla^t\phi^*) + 2e^2A^t\phi\phi^*, \\ &= 2e(\phi_1\partial_t\phi_2 - \phi_2\partial_t\phi_1) - 2e^2A_t(\phi_1^2+\phi_2^2), \\ &= 2e(\phi_1\Pi_{\phi 2} - \phi_2\Pi_{\phi 1}) - 2e^2A_t(\phi_1^2+\phi_2^2). \end{aligned} \tag{66}$$

Via (59) and (22) we thereby obtain

$$\rho = 2e(\phi_1\Pi_{\phi 2} - \phi_2\Pi_{\phi 1}) - 2e^2A_t(\phi_1^2+\phi_2^2). \tag{67}$$

Then, (67) into (65) yields

$$\nabla^i\nabla_i U = -(2e(\phi_1\Pi_{\phi 2} - \phi_2\Pi_{\phi 1}) + 2e^2A_t(\phi_1^2+\phi_2^2)), \tag{68}$$

a second order initial value condition. Having specified A_t, A_r, ϕ_1, ϕ_2 (and thus Ω_{At}, Ω_{Ar}, $\Omega_{\phi 1}$, $\Omega_{\phi 2}$), and $\Pi_{\phi 1}$ and $\Pi_{\phi 2}$, and solved for U, we proceed to

[4]Having established this procedure for electromagnetism, Piran and York [9] then generalize it to gravitational initial data problem via decomposition of the conformal extrinsic curvature tensor. The reader is referred to their paper for details on this method, commonly known as the *York procedure*.

determine the remaining two fields—Π_{At} and Π_{Ar}—at the initial time. Using (62) we will solve for E_r. Once we have determined E_r, we will use the result in the definition of the electromagnetic field-strength tensor to solve for Π_{Ar}. Explicitly, Π_{Ar} will be given by

$$
\begin{aligned}
E_r &= F_{rt} \\
&= \Omega_{At} - \Pi_{Ar}.
\end{aligned} \tag{69}
$$

Then, having determined Π_{Ar}, we determine Π_{At} via the Lorentz gauge condition,

$$
\Pi_{At} = \Omega_{Ar} + 2r^{-1}A_r. \tag{70}
$$

3 Numerical Solution

Although it is not the only numerical method that could be used to solve the flat, spherically symmetric, massive, complex scalar Cauchy problem, we will use finite difference (FD) methods.[5] We will use second order accurate differencing approximations (FDA's), on a temporally and spatially uniform mesh.[6] Although higher order schemes will be more accurate for a given grid spacing, they are numerically more demanding and not necessary for the focus of the present study.

Having recast our full set of equations (38) through (41) and (43) through (50) in second order finite difference form, we use the Rapid Numerical Prototyping (RNPL) language[7] to generate our FORTRAN code and solve the

[5] I have found [6], [7] and [8] particularly informative background reading.

[6] Adaptive mesh refinement may become necessary at some point of future study.

[7] [1] is a good reference for RNPL.

equations using the iterative Crank-Nicolson technique. The initial data constraint (68) is solved using LAPACK's DGTSV tridiagonal solver.

3.1 RNPL Code

In our RNPL source code we have implemented the second order FDA's to (38) through (41) and (43) through (50). (See *Appendix A.*) Although iterative Crank-Nicolson is used, we have also included the second order leapfrog derivative operators since they may be useful for independent residual testing in later study. Also observe that we have implemented Kreiss-Oliger dissipation—it was observed that the code is quite unstable without it, and fails soon after the fields encounter the inner ($r = 0$) boundary.

We have implemented the forward differencing in the equations of motion for $\Pi_{\phi 1}$, $\Pi_{\phi 2}$ and Π_{At} at $r = 0$. Also, we set Π_{Ar} equal to zero, since A_r must always vanish at origin of coordinates. Because of regularity, we have set $\Omega_{\phi 1}$, $\Omega_{\phi 2}$ and Ω_{At} to zero, but place no constraint on Ω_{Ar}. Since the Kreiss-Oliger dissipation is not operative on the grid point next to the $r = 0$ boundary, we tried applying interpolation schemes on the fields to dampen high frequency features. However, it was found that this only led to instabilities, and so the condition was relaxed. In any case, it was found that implementing the standard (centered, non-dissipative) equations of motion on this next-to-boundary grid point resulted in a smooth, well behaved evolution.

Now, as mentioned previously, we are wanting all information reaching the $r = r_{max}$ outer boundary to leave the computational domain. We have thus created a filter by applying the approximate (Sommerfeld) outgoing radiation condition to the last five grid points for fields $\Pi_{\phi 1}$, $\Pi_{\phi 2}$, Π_{At} and Π_{Ar}. As can be clearly seen in the MPEG format video clips at http://laplace.physics.ubc.ca/Students/petryk/index.html, this filtering was stable and apparently transmitted all information reaching $r = r_{max}$.

Finally, we have included a listing of the initial constraint solver in *Appendix B*.

3.2 Results

Our initializations for A_t, A_r, ϕ_1 and ϕ_2 are Gaussian. Arbitrarily, we have chosen identical initializations for ϕ_1 and A_t, and identical initializations for ϕ_2 and A_r. Also, we have imposed time-symmetric initial conditions on ϕ_1 and ϕ_2 by setting $\Pi_{\phi 1}$ and $\Pi_{\phi 2}$ identical to zero at the initial time step. In this article, we will focus on data whose initial parameters are those listed in *Appendix C*.[8] Specifically, the initial Gaussians for A_t, A_r, ϕ_1 and ϕ_2 will all be identically centered at $r = 50.0$, will have width 3.0, and amplitude 1.0. Our zero computational domain will have coordinate range $r = 0.0$ to $r_{max} = 100.0$. Our complex field will have mass $m = 1.0$, and be subject to coupling of strength $e = 0.05$.

Now, since our *level-0* grid is defined to have $nr_0 = 101$ grid points, a *level-n* grid will have

$$nr_n = 100 \cdot 2^n + 1 \tag{71}$$

discrete points. Our representative *level-2* grid will thus have a total of $nr_2 =$ 400 points. Furthermore, we will use a dissipation factor of 0.5, Courant factor $\lambda = 0.2$, and our system will evolve for a total of 1000 dynamical time steps.

[8] However, the solution for a second choice of initial parameters is displayed in MPEG format video clips fount at http://laplace.physics.ubc.ca/Students/petryk/index.html.

Evolutions for *level-0* through *level-4* proceed in a stable manner completion for the representative initial data. Higher level evolutions were not fully tested. Fields ϕ_1, ϕ_2, A_t and A_r evolve as shown in Figure 1 through Figure 4 (also shown at http://laplace.physics.ubc.ca/Students/petryk/index.html).

Since the coupling constant e is sufficiently large, the fields display noticeably different evolution by $t = 60$.[9] By $t = 180$, fields ϕ_1 and ϕ_2 are oscillating out of phase, and have dramatically different amplitudes. Furthermore, these fields tend to linger near $r = 0$ longer than in the case of decoupled massive Klein-Gordon fields. Now, as seen in the figures and MPEG's, a significant portion of A_t and A_r also linger near the origin of spherical symmetry for an extended period of time. As well, it is clear that the outgoing boundary condition filter works as intended and apparently transmits all fields reaching $r = r_{max}$ (most clearly seen in MPEG's). As mentioned, a set of MPEG's for evolution from different initial data is available at http://laplace.physics.ubc.ca/Students/petryk/index.html.

3.3 Tests - Convergence, Energy and Charge Conservation

For some solution to our FDA we can define its *convergence factor* as follows. Take the l_2-norm of the difference between solutions at *level-(n+1)* and *level-n*. Do likewise for the solutions at *level-(n+2)* and *level-(n+1)*. Then, the ratio of the first l_2-norm to the second l_2-norm is our convergence factor. If our finite difference scheme is second-order accurate, we expect a convergence factor near 4. We have calculated such convergence factors for ϕ_1, ϕ_2, A_t

[9] If the the coupling constant e were set to zero, ϕ_1 and ϕ_2 would evolve as identical massive Klein-Gordon fields.

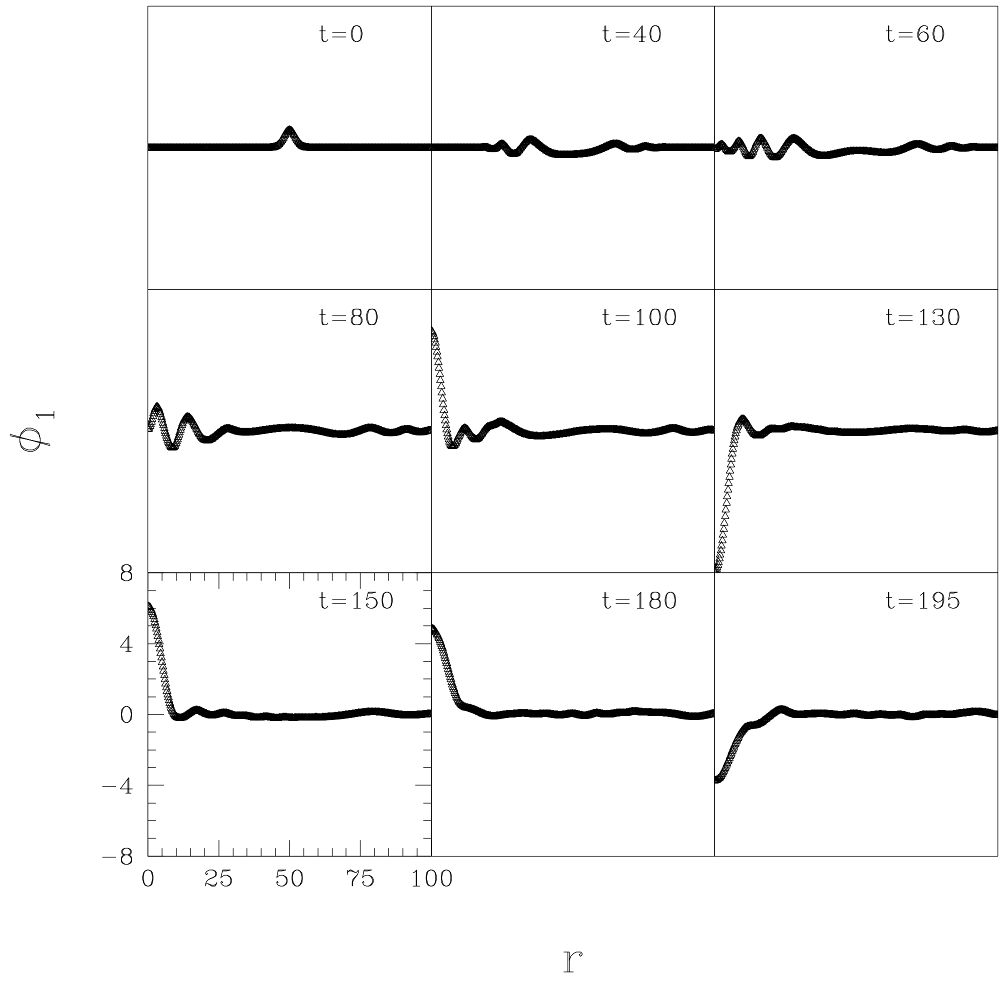

Figure 1: The evolution of ϕ_1 in time.

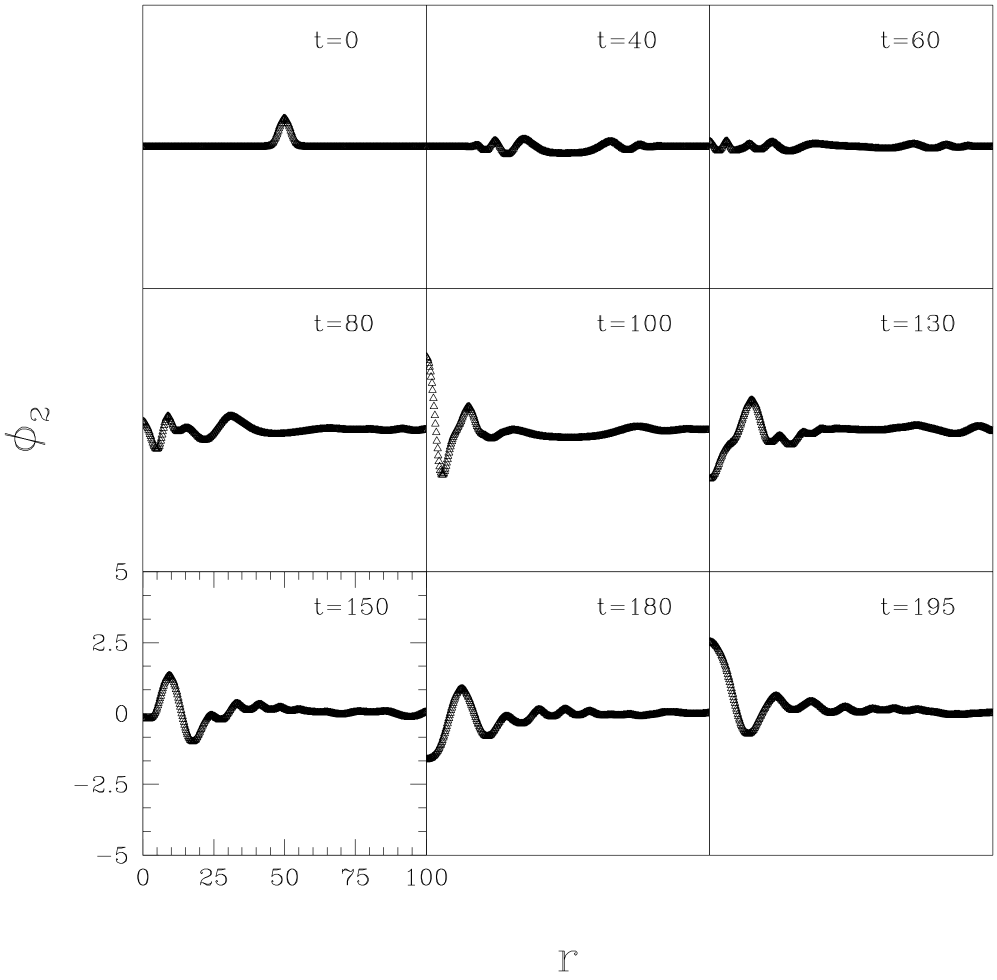

Figure 2: The evolution of ϕ_2 in time.

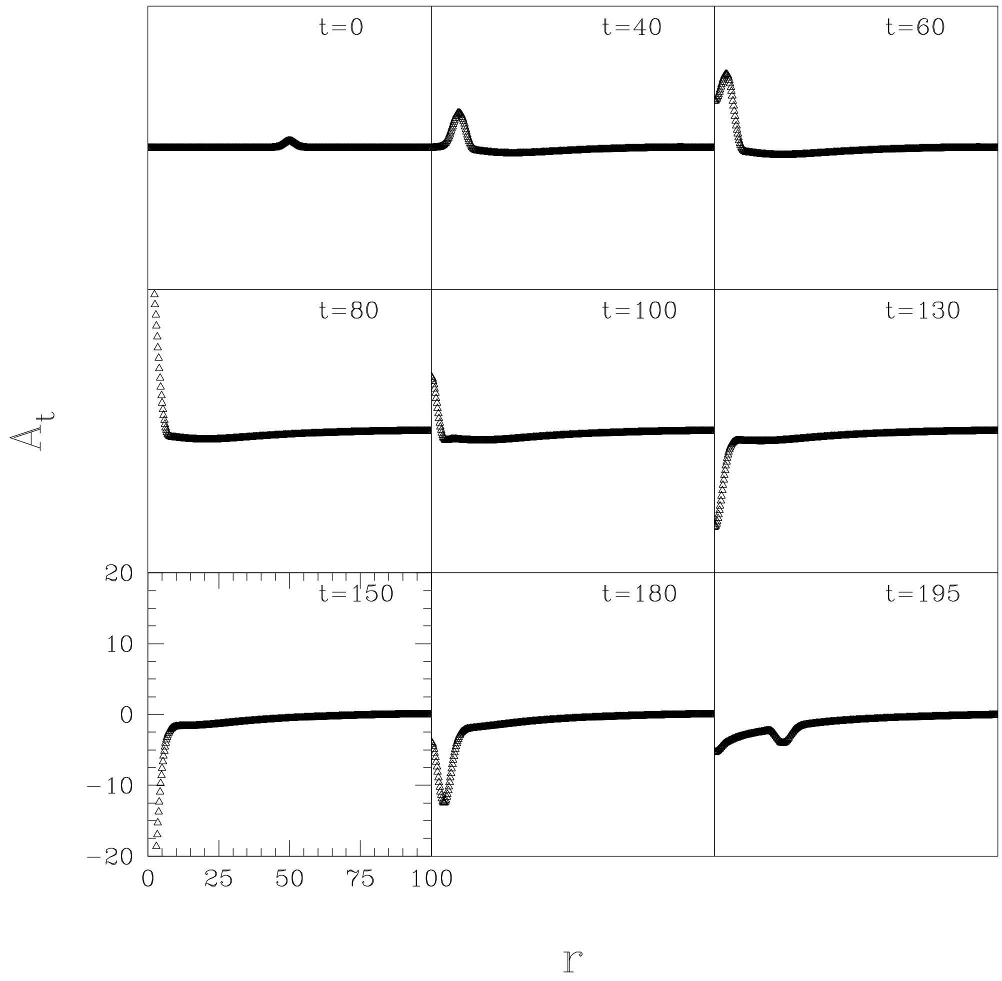

Figure 3: The evolution of A_t in time.

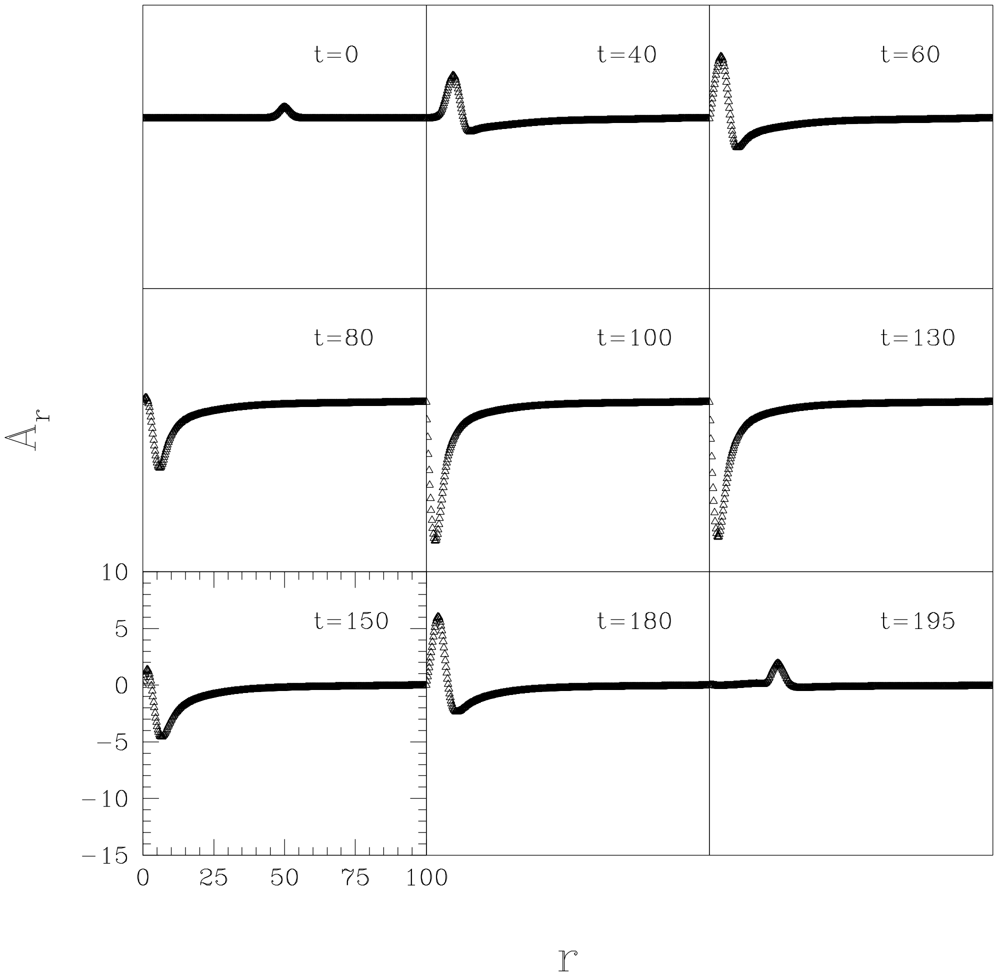

Figure 4: The evolution of A_r in time.

and A_r from our solutions at *level-4*, *level-3* and *level-2*. They are plotted in Figure 5 through Figure 8 as functions of time. Clearly, each of the solutions display convergence factors near 4 for $t < 45$. Near $t = 45$ the fields encounter the inner boundary. Now, while ϕ_1 and ϕ_2 remain rapidly oscillating near the boundary after $t = 45$, A_t and A_r do not display such behaviour. Thus, while the oscillations in ϕ_1 and ϕ_2 cause high frequency oscillation in their convergence factors, the convergence factors of A_t and A_r remain reasonably well-behaved after the initial encounter. On average, however, all four convergence factors are near or above 4—an indication that we are, in fact, approaching a continuum solution as the level of discretization increases.

Although we approach a continuum solution, it is not guaranteed we are converging towards the *correct* solution. This is why we compute the total energy of the system. For the given Lagrangian (4), the stress energy tensor is

$$\begin{aligned}
T_{\mu\nu} &= -2\frac{\partial L}{\partial g^{\mu\nu}} + g_{\mu\nu}L, \\
&= 2(\partial_\mu\phi)(\partial_\nu\phi^*) - 2ie(A_\mu\phi(\partial_\nu\phi^*) - A_\nu(\partial_\mu\phi)\phi^*) + 2e^2A_\mu A_\nu\phi\phi^* \\
&\quad +F_{\mu\sigma}F_{\nu\tau}g^{\sigma\tau} \\
&\quad +\Big((-(\partial_\sigma\phi)(\partial_\tau\phi^*))g^{\tau\sigma} + ie(A_\sigma\phi(\partial_\tau\phi^*)g^{\tau\sigma} - A_\sigma(\partial_\tau\phi)\phi^*g^{\tau\sigma}) - e^2A_\sigma A_\tau g^{\tau\sigma}\phi\phi^*\Big)\,g_{\mu\nu} \\
&\quad -\frac{1}{4}F_{\sigma\tau}F^{\sigma\tau}g_{\mu\nu} - m^2\phi\phi^*g_{\mu\nu}
\end{aligned} \tag{72}$$

The energy of the system is then

$$\begin{aligned}
T_{00} &= \Pi_{\phi1}^2 + \Pi_{\phi2}^2 + \Omega_{\phi1}^2 + \Omega_{\phi1}^2 + m^2(\phi_1^2 + \phi_2^2) \\
&\quad +2eA_t(\phi_2\Pi_{\phi1} - \phi_1\Pi_{\phi2}) + 2eA_r(\phi_2\Omega_{\phi1} - \phi_1\Omega_{\phi2}) \\
&\quad +e^2(A_t^2 + A_r^2)(\phi_1^2 + \phi_2^2) \\
&\quad +\frac{1}{2}\left(\Pi_{Ar}^2 - 2\Pi_{Ar}\Omega_{At} + \Omega_{At}^2\right)
\end{aligned} \tag{73}$$

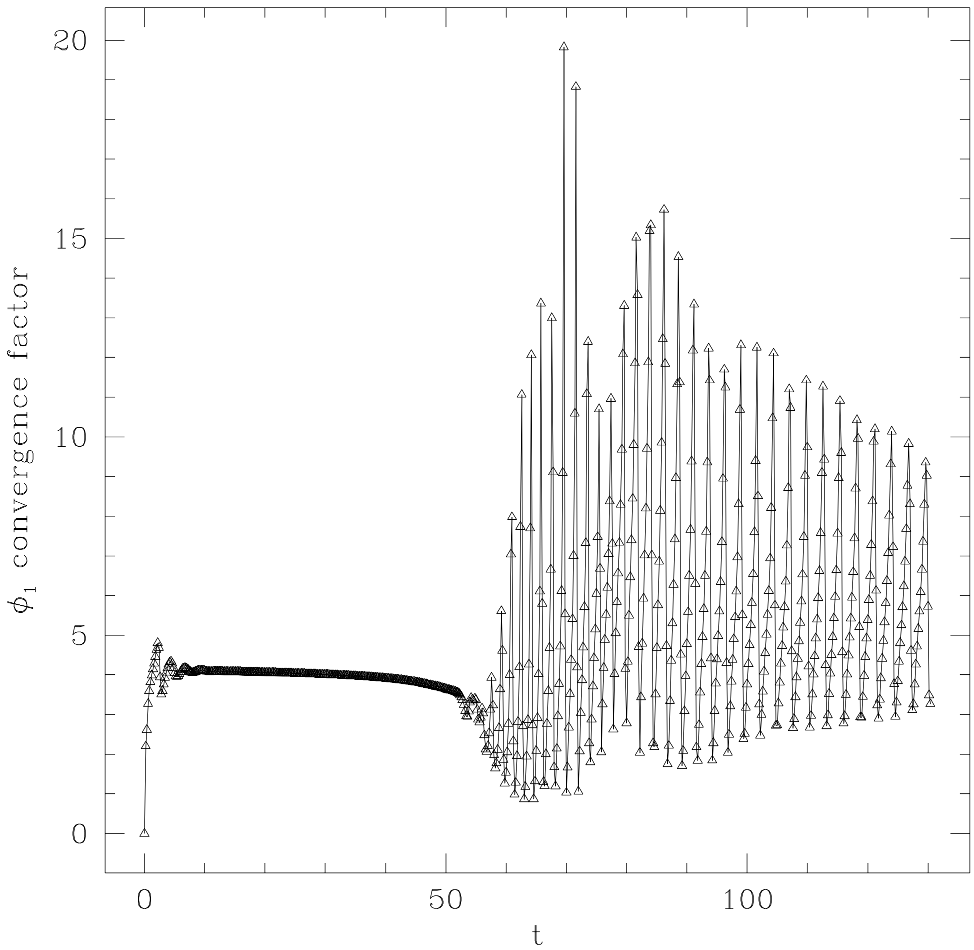

Figure 5: The convergence factor for ϕ_1 as a function of time.

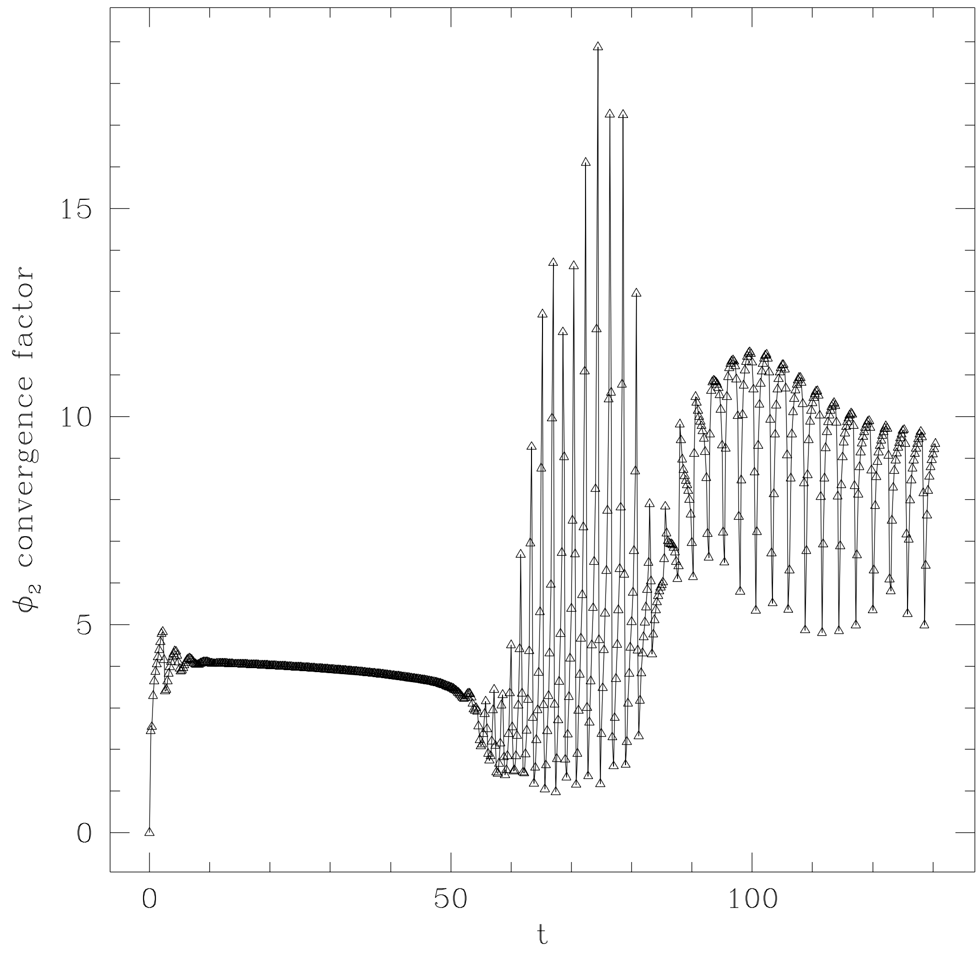

Figure 6: The convergence factor for ϕ_2 as a function of time.

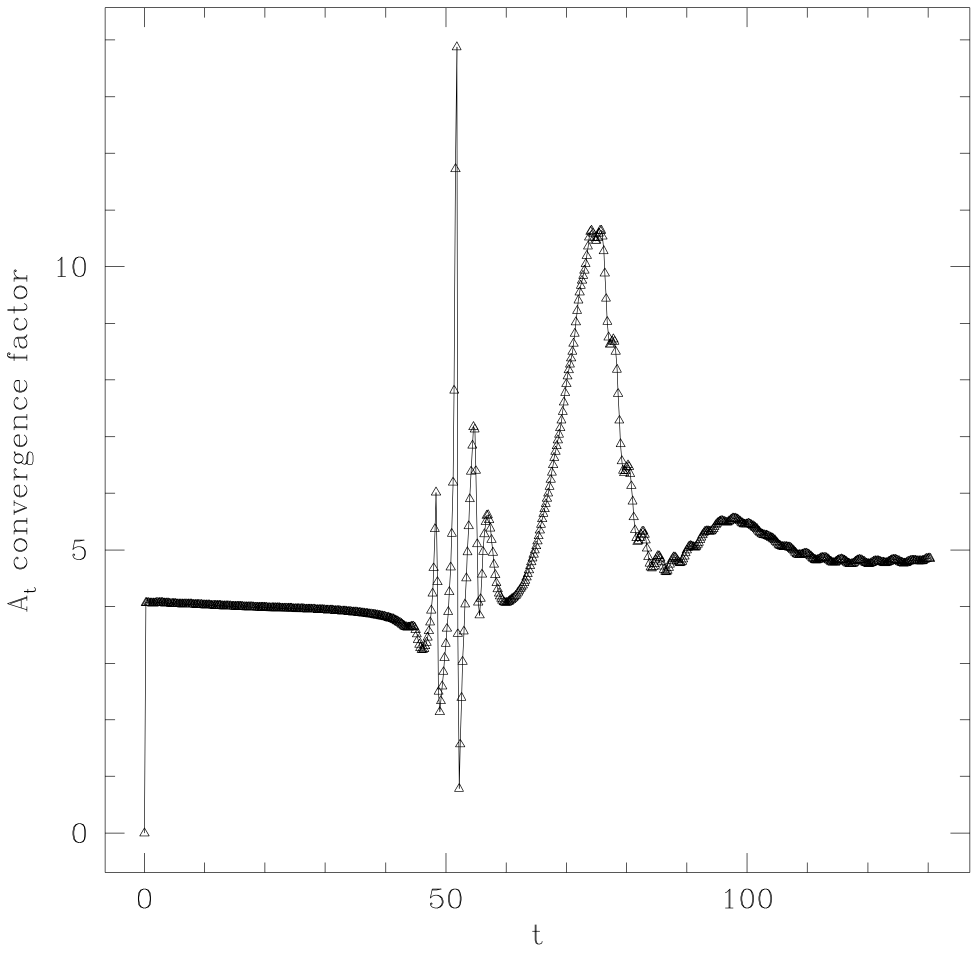

Figure 7: The convergence factor for A_t as a function of time.

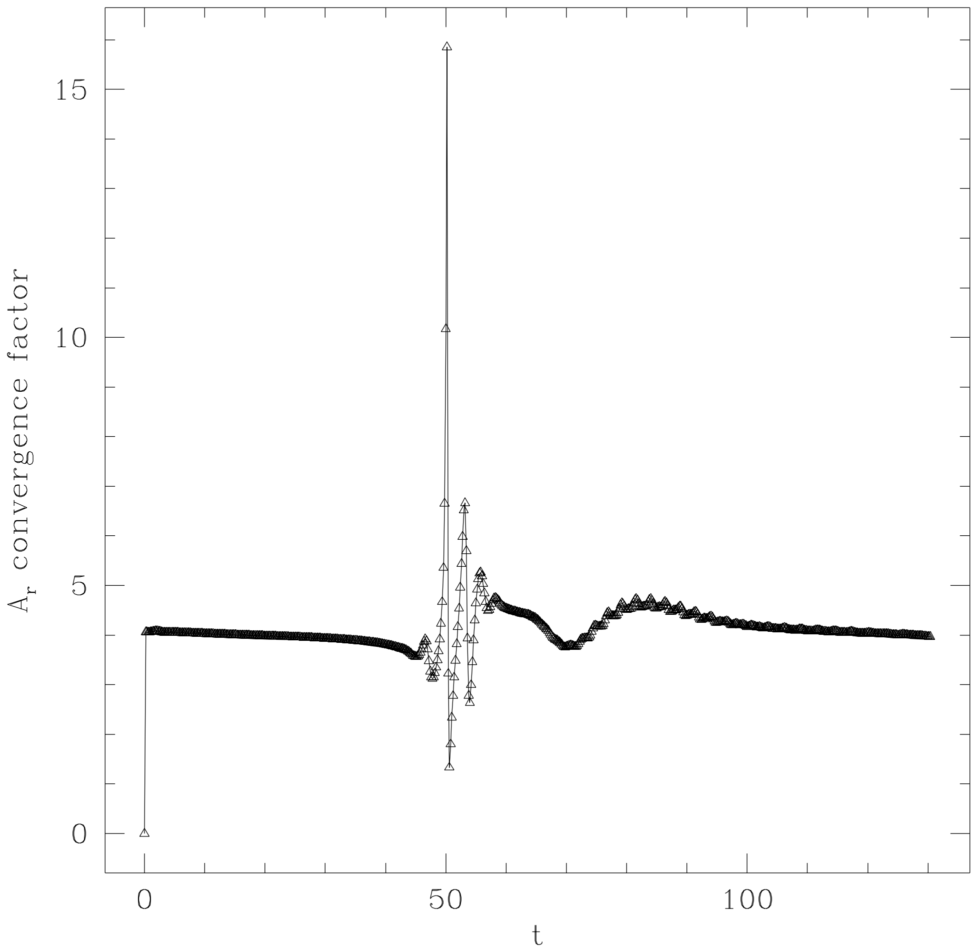

Figure 8: The convergence factor for A_r as a function of time.

In *Appendix D* we have included a routine which solves for this energy.[10] In Figure 9 we plot the total energy contained within $r = r_{max}$ as a function of time. Clearly, as we go to higher discretization levels, the energy converges to a conserved quantity. Figure 10 shows the convergence in detail. After time $t = 55$, energy dissipation is observed as the fields leave the domain of computation at the outer boundary.

As a second test of the code, we could calculate the total charge of the system. If all dynamics are well within $r = r_{max}$ we would expect the total charge to be conserved until information propagates to the outer boundary. At that point we would expect the total charge contained within $r = r_{max}$ to decrease in time.[11] Now, from (66) and (22) in spherical symmetry, we obtain

$$\begin{aligned} Q &= \int \rho d^3x \\ &= \int \left(2e(\phi_1\Pi_{\phi 2} - \phi_2\Pi_{\phi 1}) - 2e^2 A_t(\phi_1^2 + \phi_2^2)\right) r^2 sin\theta dr d\theta d\phi, \\ &= 4\pi \int \left(2e(\phi_1\Pi_{\phi 2} - \phi_2\Pi_{\phi 1}) - 2e^2 A_t(\phi_1^2 + \phi_2^2)\right) r^2 dr, \end{aligned} \tag{74}$$

where the remaining integral in r is over the entire computational domain. Figure 11 shows the total charge contained within $r = r_{max}$ as a function of time. Clearly, for our choice of initial data, we do not observe charge conservation. In fact, the charge seems to converge to an obviously dissipative state. This situation clearly deserves further investigation. We believe that the routine used to calculate total charge (see *Appendix F*) is in error.

[10]When coupling constant e is set to zero, the energy of the complex scalar system (73) reduces to the energy of two decoupled Klein-Gordon fields. *Appendix E* includes a listing for the routine which solves for the Klein-Gordon field contributions to the total energy of the system.

[11]It was observed in [3] that the total charge for a certain set of initial conditions of the axisymmetric complex scalar system obeyed the expected conservation law.

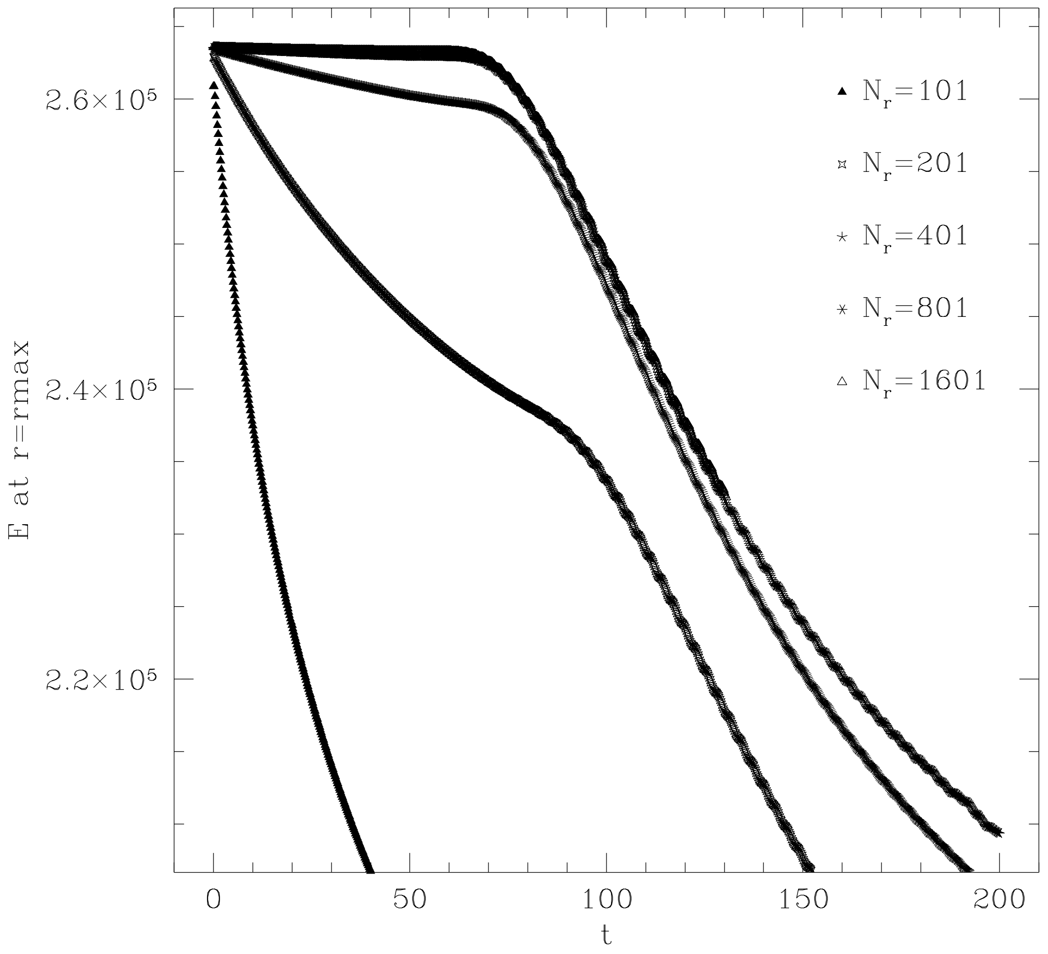

Figure 9: The total energy of the system converges towards a conserved quantity.

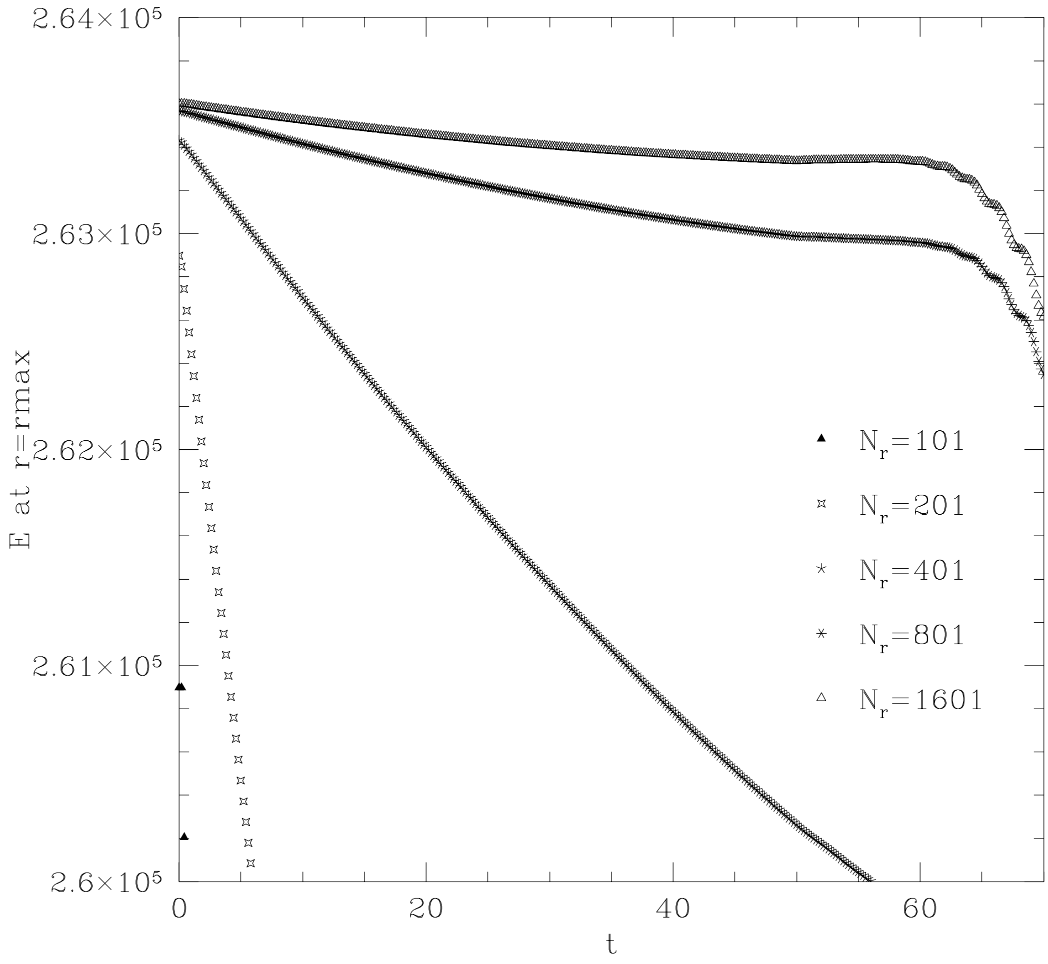

Figure 10: The total energy of the system converges towards a conserved quantity (detail).

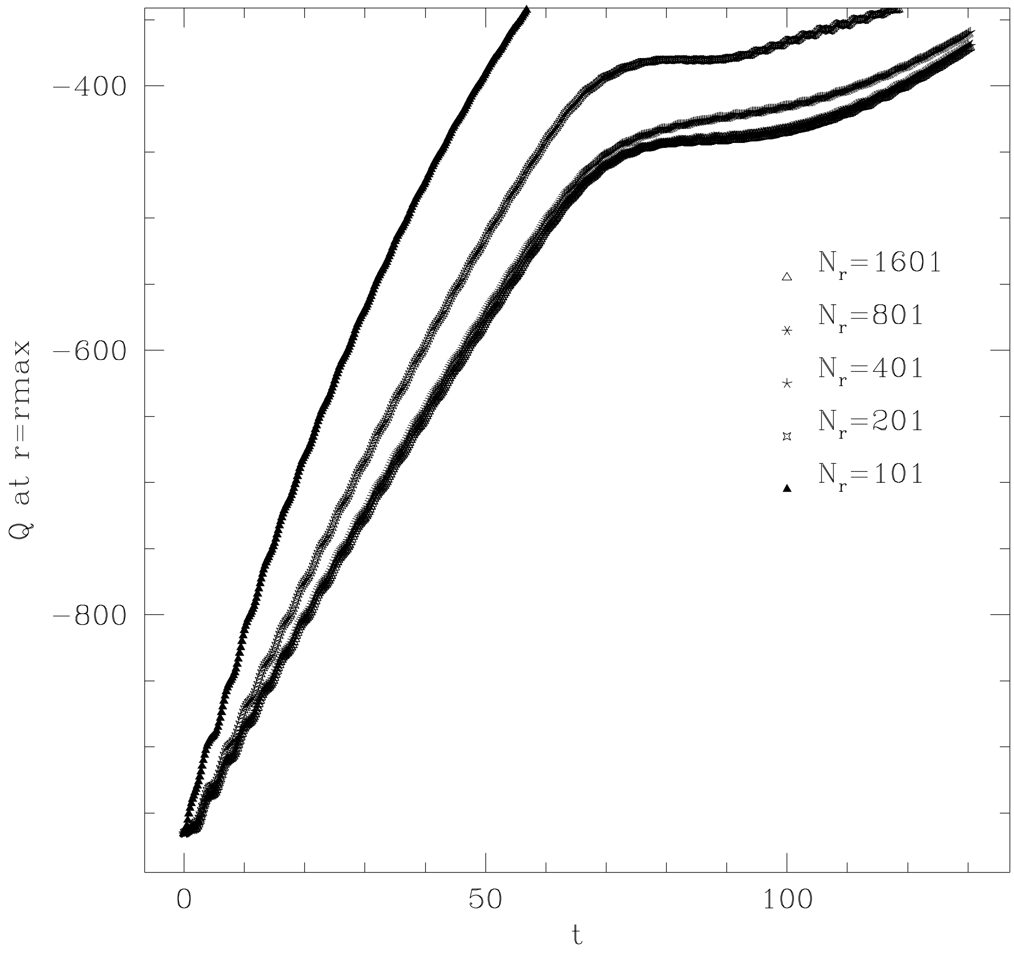

Figure 11: The total charge of the system is apparently not conserved for the given initial data.

Other tests which could be implemented to check our code are: (1) independent residual evaluation; (2) Lorentz gauge condition preservation; and (3) preservation of Gauss' law.

4 Conclusions

We have successfully solved for the dynamics of the complex scalar field coupled to electromagnetism. Our fields display the expected convergence properties, and the energy of the system approaches conservation. However, the expected total charge conservation is not observed. Further investigation and testing would be useful.

5 Acknowledgements

R.P. would like to thank Inaki Olabarrietta for fruitful discussions and advice during investigation. R.P. also thanks Matthew Choptuik for his advice and suggestions.

A RNPL Code

Source file: EM_rnpl

```
######################################################################
# RNPL source code for the spherically symmetric electromagnetically
# coupled complex scalar field
#
# Phi = Phi1 + i*Phi2
#
# in flat spacetime with spherical polar coordinates. Written in first
# order form, the equations of motion are
#
# PiAt_t = OmegaAt_r
#        + (2/r)*OmegaAt
#        + 2*e*(Phi1*PiPhi2 - Phi2*PiPhi1)
#        - 2*(e**2)*At*(Phi1**2 + Phi2**2),
#
# PiAr_t = OmegaAr_r
#        + (2/r)*OmegaAr
#        - (2/(r**2))*Ar
#        + 2*e*(Phi1*OmegaPhi2 - Phi2*OmegaPhi1)
#        - 2*(e**2)*Ar*(Phi1**2 + Phi2**2),
#
# PiPhi1_t = OmegaPhi1_r
#          + (2/r)*OmegaPhi1
#          - (m**2)*Phi1
#          - 2*e*(PiPhi2*At - OmegaPhi2*Ar)
#          - (e**2)*(-At**2 + Ar**2)*Phi1,
#
# PiPhi2_t = OmegaPhi2_r
#          + (2/r)*OmegaPhi2
#          - (m**2)*Phi2
#          - 2*e*(-PiPhi1*At + OmegaPhi1*Ar)
#          - (e**2)*(-At**2 + Ar**2)*Phi2,
#
# and
#
# PiAt = OmegaAr + 2*(r**(-1))*Ar
#
# is the Lorentz gauge condition, where
#
# PiAt := At_t
# PiAr := Ar_t
# PiPhi1 := Phi1_t
# PiPhi2 := Phi2_t
# OmegaAt := At_r
# OmegaAr := Ar_r
# OmegaPhi1 := Phi1_r
# OmegaPhi2 := Phi2_r
#
# using Crank-Nicolson with (implicit) Kreiss-Oliger dissipation.
#
# Last modified by Roman Petryk, Department of Physics and Astronomy,
```

```
# University of British Columbia, on 12 December, 2000.
########################################################################

# Set the memory size (only needed for FORTRAN)
system parameter int memsiz := 1000000




########################################################################
# Begin by specifying the parameters.
#
# Parameter values not specified here are given in the idx
# (x=0,1,2,...) initial data file.
########################################################################

parameter float rmax
parameter float rmin
parameter float m
parameter float e
parameter float amp1
parameter float amp2
parameter float rcen1
parameter float rcen2
parameter float rwid1
parameter float rwid2
parameter float epsdis




########################################################################
# Now define the coordinate system with time coordinate t and spatial
# coordinate r.
########################################################################

sph coordinates t, r




########################################################################
# Define the uniform spherical grid with index region [1:Nr] and
# spatial range {rmin:rmax}.
########################################################################

uniform sph grid g1 [1:Nr] {rmin:rmax}
```

```
##########################################################################
# Define grid functions Pi, Phi and phioKG on g1 at times 0,1.
##########################################################################

float At on g1 at 0,1 {out_gf := 1}
float Ar on g1 at 0,1 {out_gf := 1}
float Phi1 on g1 at 0,1 {out_gf := 1}
float Phi2 on g1 at 0,1 {out_gf := 1}

float PiAt on g1 at 0,1 {out_gf := 1}
float PiAr on g1 at 0,1 {out_gf := 1}
float PiPhi1 on g1 at 0,1 {out_gf := 1}
float PiPhi2 on g1 at 0,1 {out_gf := 1}

float OmegaAt on g1 at 0,1 {out_gf := 1}
float OmegaAr on g1 at 0,1 {out_gf := 1}
float OmegaPhi1 on g1 at 0,1 {out_gf := 1}
float OmegaPhi2 on g1 at 0,1 {out_gf := 1}

float cnst on g1 at 0 {out_gf := 1}
# Grid function for the charge density.
float rho on g1 at 0,1 {out_gf := 1}
# Grid function for the charge.
float Q on g1 at 0 {out_gf := 1}
# Grid function for the energy of the Klein Gordon fields.
float Ekg on g1 at 0 {out_gf := 1}
# Grid function for the total energy of the system.
float Es on g1 at 0 {out_gf := 1}




##########################################################################
# Now define the necessary Crank_Nicolson derivatives.
##########################################################################

# Crank-Nicolson time derivative. Derivative is O(h^2) when centered
# at t=1/2.
operator D_CNT(f,t) := (<1>f[0]-<0>f[0])/dt

# O(h^2) t=0, r=0 centered spatial derivative.
operator D_CNR(f,r) := (<0>f[1]-<0>f[-1])/(2*dr)

# O(h^2) t=0, r=0 centered spatial derivative for future time.
operator D_CNRF(f,r) := (<1>f[1]-<1>f[-1])/(2*dr)

# O(h^2) t=0, r=0 centered forward difference spatial derivative.
operator D_CNFWR(f,r) := (-3*<0>f[0]+4*<0>f[1]-<0>f[2])/(2*dr)

# O(h^2) t=0, r=0 centered forward difference spatial derivative for
# future time.
operator D_CNFWRF(f,r) := (-3*<1>f[0]+4*<1>f[1]-<1>f[2])/(2*dr)
```

```
# O(h^2) t=0, r=0 centered backward difference spatial derivative.
operator D_CNBWR(f,r) := (3*<0>f[0]-4*<0>f[-1]+<0>f[-2])/(2*dr)

# O(h^2) t=0, r=0 centered backward difference spatial derivative for
# future time.
operator D_CNBWRF(f,r) := (3*<1>f[0]-4*<1>f[-1]+<1>f[-2])/(2*dr)

# Forward time averaging operator---necessary because while for
# Crank-Nicolson we are evaluating at times t=1/2 we want to view
# t=1.
operator AVGT(f,t) := (<1>f[0]+<0>f[0])/2

operator AVGTD(f,t) := (<1>f[0]+<0>f[0])/2

# Dissipation operator.
operator DISS(f,t) := ((-epsdis)/(16*dt))*(6*<0>f[0]+<0>f[2]
                        +<0>f[-2]-4*(<0>f[1]+<0>f[-1]))

# Dissipation operator for future time..
operator DISSF(f,t) := ((-epsdis)/(16*dt))*(6*<1>f[0]+<1>f[2]
                         +<1>f[-2]-4*(<1>f[1]+<1>f[-1]))



#######################################################################
# Now define the necessary leapfrog derivatives.
#######################################################################

# O(h^2) t=0, r=0 centered first time derivative.
operator D_LFT(f,t) := (<1>f[0] - <-1>f[0])/(2*dt)

# O(h^2) t=0, r=0 second time derivative.
operator D_LFTT(f,t,t) := (<1>f[0] - 2*<0>f[0] + <-1>f[0])/(dt*dt)

# O(h^2) t=0, r=0 centered first spatial derivative.
operator D_LFR(f,r) := (<0>f[1] - <0>f[-1])/(2*dr)

# O(h^2) t=0, r=0 centered forward difference spatial derivative.
operator D_LFFWR(f,r) := (-3*<0>f[0] + 4*<0>f[1] - <0>f[2])/(2*dr)

# O(h^2) t=0, r=0 centered backward difference spatial derivative.
operator D_LFBWR(f,r) := (3*<0>f[0] - 4*<0>f[-1] + <0>f[-2])/(2*dr)



#######################################################################
# Define the residuals.
#######################################################################
```

```
evaluate residual PiPhi1 {

# Forward differencing equation of motion. Employ L'Hopital's rule to
# get rid of the indeterminacy of the (2/r)*OmegaPhi1 term at r=0.
    [1:1] := D_CNT(PiPhi1,t)
                 - AVGT( D_CNFWR(OmegaPhi1,r)
                 + 2*D_CNFWR(OmegaPhi1,r)
                 - (m^2)*Phi1
                 - 2*e*(+PiPhi2*At - OmegaPhi2*Ar)
                 - (e^2)*(-At^2 + Ar^2)*Phi1, t);

# The equation of motion.
[2:2] := D_CNT(PiPhi1,t)
                 - AVGT( D_CNR(OmegaPhi1,r)
                 + (2/r)*OmegaPhi1
                 - (m^2)*Phi1
                 - 2*e*(PiPhi2*At - OmegaPhi2*Ar)
                 - (e^2)*(-At^2 + Ar^2)*Phi1, t);
# Apply quadratic interpolation to determine PiPhi at the second spatial
# point.
# [2:2] := <1>PiPhi1[0]
#                 - (1/4)*( 3*(<1>PiPhi1[-1]) + <1>PiPhi1[1] );

# The equation of motion with dissipation.
[3:Nr-5] := D_CNT(PiPhi1,t)
                 - AVGTD( DISS(PiPhi1,t), t)
                 - AVGT( D_CNR(OmegaPhi1,r)
                 + (2/r)*OmegaPhi1
                 - (m^2)*Phi1
                 - 2*e*(PiPhi2*At - OmegaPhi2*Ar)
                 - (e^2)*(-At^2 + Ar^2)*Phi1, t);

# The equation of motion.
# [Nr-1:Nr-1] := D_CNT(PiPhi1,t)
#                 - AVGT( D_CNR(OmegaPhi1,r)
#                 + (2/r)*OmegaPhi1
#                 - (m^2)*Phi1
#                 - 2*e*(PiPhi2*At - OmegaPhi2*Ar)
#                 - (e^2)*(-At^2 + Ar^2)*Phi1, t);

# The field Phi1 remains constant at rmax.
# [Nr:Nr] := <1>PiPhi1[0];
# Approximate Sommerfeld outgoing radiation condition filter.
[Nr-4:Nr] := <1>PiPhi1[0]
                 + (1/r)*(<1>Phi1[0])
                 + <1>OmegaPhi1[0];
# The equation of motion.
# [Nr:Nr] := D_CNT(PiPhi1,t)
#                 - AVGT( D_CNBWR(OmegaPhi1,r)
#                 + (2/r)*OmegaPhi1
#                 - (m^2)*Phi1
#                 - 2*e*(PiPhi2*At - OmegaPhi2*Ar)
#                 - (e^2)*(-At^2 + Ar^2)*Phi1, t);
```

```
}


evaluate residual PiPhi2 {

# Forward differencing equation of motion. Employ L'Hopital's rule to
# get rid of the indeterminacy of the (2/r)*OmegaPhi2 term at r=0.
[1:1] := D_CNT(PiPhi2,t)
                 - AVGT( D_CNFWR(OmegaPhi2,r)
                 + 2*D_CNFWR(OmegaPhi2,r)
                 - (m^2)*Phi2
                 - 2*e*(-PiPhi1*At + OmegaPhi1*Ar)
                 - (e^2)*(-At^2 + Ar^2)*Phi2, t);

# The equation of motion.
[2:2] := D_CNT(PiPhi2,t)
                 - AVGT( D_CNR(OmegaPhi2,r)
                 + (2/r)*OmegaPhi2
                 - (m^2)*Phi2
                 - 2*e*(-PiPhi1*At + OmegaPhi1*Ar)
                 - (e^2)*(-At^2 + Ar^2)*Phi2, t);
# Apply quadratic interpolation to determine PiPhi2 at the second spatial
# point.
# [2:2] := <1>PiPhi2[0]
#                  - (1/4)*( 3*(<1>PiPhi2[-1]) + <1>PiPhi2[1] );

# The equation of motion with dissipation.
[3:Nr-5] := D_CNT(PiPhi2,t)
                 - AVGTD( DISS(PiPhi2,t), t)
                 - AVGT( D_CNR(OmegaPhi2,r)
                 + (2/r)*OmegaPhi2
                 - (m^2)*Phi2
                 - 2*e*(-PiPhi1*At + OmegaPhi1*Ar)
                 - (e^2)*(-At^2 + Ar^2)*Phi2, t);

# The equation of motion.
# [Nr-1:Nr-1] := D_CNT(PiPhi2,t)
#                  - AVGT( D_CNR(OmegaPhi2,r)
#                  + (2/r)*OmegaPhi2
#                  - (m^2)*Phi2
#                  - 2*e*(-PiPhi1*At + OmegaPhi1*Ar)
#                  - (e^2)*(-At^2 + Ar^2)*Phi2, t);

# The field Phi2 remains constant at rmax.
# [Nr:Nr] := <1>PiPhi2[0];
# Approximate Sommerfeld outgoing radiation condition filter.
[Nr-4:Nr] := <1>PiPhi2[0]
                 + (1/r)*(<1>Phi2[0])
                 + <1>OmegaPhi2[0];
# [Nr:Nr] := D_CNT(PiPhi2,t)
#                  - AVGT( D_CNBWR(OmegaPhi2,r)
#                  + (2/r)*OmegaPhi2
#                  - (m^2)*Phi2
#                  - 2*e*(-PiPhi1*At + OmegaPhi1*Ar)
```

```
#                  - (e^2)*(-At^2 + Ar^2)*Phi2, t);

}


evaluate residual OmegaPhi1 {

# Phi1 is regular at r=0.
[1:1] := <1>OmegaPhi1[0];

# Time and space differentiation commute.
[2:2] := D_CNT(OmegaPhi1,t)
                 - AVGT( D_CNR(PiPhi1,r), t);
# Apply linear interpolation to determine OmegaPhi1 at the second spatial
# point.
# [2:2] := <1>OmegaPhi1[0]
#                  - (1/2)*(<1>OmegaPhi1[-1] + <1>OmegaPhi1[1]);

# Time and space differentiation commute, plus dissipation.
[3:Nr-2] := D_CNT(OmegaPhi1,t)
                 - AVGTD( DISS(OmegaPhi1,t), t)
                 - AVGT( D_CNR(PiPhi1,r), t);

# Time and space differentiation commute.
[Nr-1:Nr-1] := D_CNT(OmegaPhi1,t)
                 - AVGT( D_CNR(PiPhi1,r), t);

# Time and space differentiation commute.
[Nr:Nr] := D_CNT(OmegaPhi1,t)
                 - AVGT( D_CNBWR(PiPhi1,r), t);

}


evaluate residual OmegaPhi2 {

# Phi2 is regular at r=0.
[1:1] := <1>OmegaPhi2[0];

# Time and space differentiation commute.
[2:2] := D_CNT(OmegaPhi2,t)
                 - AVGT( D_CNR(PiPhi2,r), t);
# Apply linear interpolation to determine OmegaPhi2 at the second spatial
# point.
# [2:2] := <1>OmegaPhi2[0]
#                  - (1/2)*(<1>OmegaPhi2[-1] + <1>OmegaPhi2[1]);

# Time and space differentiation commute, plus dissipation.
[3:Nr-2] := D_CNT(OmegaPhi2,t)
                 - AVGTD( DISS(OmegaPhi2,t), t)
                 - AVGT( D_CNR(PiPhi2,r), t);

# Time and space differentiation commute.
[Nr-1:Nr-1] := D_CNT(OmegaPhi2,t)
```

```
                 - AVGT( D_CNR(PiPhi2,r), t);

# Time and space differentiation commute.
[Nr:Nr] := D_CNT(OmegaPhi2,t)
                 - AVGT( D_CNBWR(PiPhi2,r), t);

}


evaluate residual PiAt {

# Forward differencing equation of motion. Employ L'Hopital's rule to
# get rid of the indeterminacy of the (2/r)*OmegaAt term at r=0.
[1:1] := D_CNT(PiAt,t)
                 - AVGT( D_CNFWR(OmegaAt,r)
                 + 2*D_CNFWR(OmegaAt,r)
                 + 2*e*(Phi1*PiPhi2 - Phi2*PiPhi1)
                 - 2*(e^2)*At*(Phi1^2 + Phi2^2), t);

# The equation of motion.
[2:2] := D_CNT(PiAt,t)
                 - AVGT( D_CNR(OmegaAt,r)
                 + (2/r)*OmegaAt
                 + 2*e*(Phi1*PiPhi2 - Phi2*PiPhi1)
                 - 2*(e^2)*At*(Phi1^2 + Phi2^2), t);
# Apply quadratic interpolation to determine PiAt at the second spatial
# point.
# [2:2] := <1>PiAt[0]
#                  - (1/4)*( 3*(<1>PiAt[-1]) + <1>PiAt[1] );

# The equation of motion with dissipation.
[3:Nr-5] := D_CNT(PiAt,t)
                 - AVGTD( DISS(PiAt,t), t)
                 - AVGT( D_CNR(OmegaAt,r)
                 + (2/r)*OmegaAt
                 + 2*e*(Phi1*PiPhi2 - Phi2*PiPhi1)
                 - 2*(e^2)*At*(Phi1^2 + Phi2^2), t);

# The equation of motion.
# [Nr-1:Nr-1] := D_CNT(PiAt,t)
#                  - AVGT( D_CNR(OmegaAt,r)
#                  + (2/r)*OmegaAt
#                  + 2*e*(Phi1*PiPhi2 - Phi2*PiPhi1)
#                  - 2*(e^2)*At*(Phi1^2 + Phi2^2), t);

# The field At remains constant at rmax.
# [Nr:Nr] := <1>PiAt[0];
# Approximate Sommerfeld outgoing radiation condition filter.
[Nr-4:Nr] := <1>PiAt[0]
                 + (1/r)*(<1>At[0])
                 + <1>OmegaAt[0];
# [Nr:Nr] := D_CNT(PiAt,t)
#                  - AVGT( D_CNBWR(OmegaAt,r)
#                  + (2/r)*OmegaAt
```

```
#                  + 2*e*(Phi1*PiPhi2 - Phi2*PiPhi1)
#                  - 2*(e^2)*At*(Phi1^2 + Phi2^2), t);

}


evaluate residual PiAr {

# PiAr is identically zero at r=0.
[1:1] := <1>PiAr[0];

# The equation of motion with dissipation.
[3:Nr-2] := D_CNT(PiAr,t)
                 - AVGTD( DISS(PiAr,t), t)
                 - AVGT( D_CNR(OmegaAr,r)
                 + (2/r)*OmegaAr
                 - (2/(r^2))*Ar
                 + 2*e*(Phi1*OmegaPhi2 - Phi2*OmegaPhi1)
                 - 2*(e^2)*Ar*(Phi1^2 + Phi2^2), t);

# The equation of motion.
[2:2] := D_CNT(PiAr,t)
                 - AVGT( D_CNR(OmegaAr,r)
                 + (2/r)*OmegaAr
                 - (2/(r^2))*Ar
                 + 2*e*(Phi1*OmegaPhi2 - Phi2*OmegaPhi1)
                 - 2*(e^2)*Ar*(Phi1^2 + Phi2^2), t);
# Apply linear interpolation to determine PiAr at the second spatial
# point.
# [2:2] := <1>PiAr[0]
#                  - (1/2)*( (<1>PiAr[-1]) + <1>PiAr[1] );

# The equation of motion.
[Nr-1:Nr-5] := D_CNT(PiAr,t)
                 - AVGT( D_CNR(OmegaAr,r)
                 + (2/r)*OmegaAr
                 - (2/(r^2))*Ar
                 + 2*e*(Phi1*OmegaPhi2 - Phi2*OmegaPhi1)
                 - 2*(e^2)*Ar*(Phi1^2 + Phi2^2), t);

# The field At remains constant at rmax.
# [Nr:Nr] := <1>PiAr[0];
# Approximate Sommerfeld outgoing radiation condition filter.
[Nr-4:Nr] := <1>PiAr[0]
                 + (1/r)*(<1>Ar[0])
                 + <1>OmegaAr[0];
# [Nr:Nr] := D_CNT(PiAr,t)
#                  - AVGT( D_CNBWR(OmegaAr,r)
#                  + (2/r)*OmegaAr
#                  - (2/(r^2))*Ar
#                  + 2*e*(Phi1*OmegaPhi2 - Phi2*OmegaPhi1)
#                  - 2*(e^2)*Ar*(Phi1^2 + Phi2^2), t);

}
```

```
evaluate residual OmegaAt {

# At is regular at r=0.
[1:1] := <1>OmegaAt[0];

# Time and space differentiation commute.
[2:2] := D_CNT(OmegaAt,t)
                 - AVGT( D_CNR(PiAt,r), t);
# Apply linear interpolation to determine OmegaAt at the second spatial
# point.
# [2:2] := <1>OmegaAt[0]
#                  - (1/2)*(<1>OmegaAt[-1] + <1>OmegaAt[1]);

# Time and space differentiation commute, plus dissipation.
[3:Nr-2] := D_CNT(OmegaAt,t)
                 - AVGTD( DISS(OmegaAt,t), t)
                 - AVGT( D_CNR(PiAt,r), t);

# Time and space differentiation commute.
[Nr-1:Nr-1] := D_CNT(OmegaAt,t)
                 - AVGT( D_CNR(PiAt,r), t);

# Time and space differentiation commute.
[Nr:Nr] := D_CNT(OmegaAt,t)
                 - AVGT( D_CNBWR(PiAt,r), t);

}


evaluate residual OmegaAr {

# Time and space differentiation commute.
[1:1] := D_CNT(OmegaAr,t)
                 - AVGT( D_CNFWR(PiAr,r), t);

# Time and space differentiation commute.
[2:2] := D_CNT(OmegaAr,t)
                 - AVGT( D_CNR(PiAr,r), t);
# Apply quadratic interpolation to determine OmegaAr at the second spatial
# point.
# [2:2] := <1>OmegaAr[0]
#                  - (1/4)*(3*<1>OmegaAr[-1] + <1>OmegaAr[1]);

# Time and space differentiation commute, plus dissipation.
[3:Nr-2] := D_CNT(OmegaAr,t)
                 - AVGTD( DISS(OmegaAr,t), t)
                 - AVGT( D_CNR(PiAr,r), t);

# Time and space differentiation commute.
[Nr-1:Nr-1] := D_CNT(OmegaAr,t)
                 - AVGT( D_CNR(PiAr,r), t);
```

```
# Time and space differentiation commute.
[Nr:Nr] := D_CNT(OmegaAr,t)
                 - AVGT( D_CNBWR(PiAr,r), t);

}


evaluate residual Phi1 {

# Here we include some equations which basically say the field Phi1
# is the integral of PiPhi1 with respect to time. That is, we implement
# free evolution for field Phi1.

[1:1]       := D_CNT(Phi1,t)
                 - AVGT(PiPhi1,t);

[2:2] := D_CNT(Phi1,t)
                 - AVGT(PiPhi1,t);
# Apply quadratic interpolation to determine Phi1 at the second spatial
# point.
# [2:2] := <1>Phi1[0]
#                 - (1/4)*( 3*(<1>Phi1[-1]) + <1>Phi1[1] );

# Include dissipation.
[3:Nr-2] := D_CNT(Phi1,t)
                 - AVGTD( DISS(Phi1,t), t)
                 - AVGT(PiPhi1,t);

[Nr-1:Nr-1] := D_CNT(Phi1,t)
                 - AVGT(PiPhi1,t);

[Nr:Nr] := D_CNT(Phi1,t)
                 - AVGT(PiPhi1,t);

}


evaluate residual Phi2 {

# Here we include some equations which basically say the field Phi2
# is the integral of PiPhi2 with respect to time. That is, we implement
# free evolution for Phi2.

[1:1]       := D_CNT(Phi2,t)
                 - AVGT(PiPhi2,t);

[2:2] := D_CNT(Phi2,t)
                 - AVGT(PiPhi2,t);
# Apply quadratic interpolation to determine Phi2 at the second spatial
# point.
# [2:2] := <1>Phi2[0]
#                 - (1/4)*( 3*(<1>Phi2[-1]) + <1>Phi2[1] );

# Include dissipation.
```

```
[3:Nr-2] := D_CNT(Phi2,t)
                 - AVGTD( DISS(Phi2,t), t)
                 - AVGT(PiPhi2,t);

[Nr-1:Nr-1] := D_CNT(Phi2,t)
                 - AVGT(PiPhi2,t);

[Nr:Nr] := D_CNT(Phi2,t)
                 - AVGT(PiPhi2,t);

}


evaluate residual At {

# Here we include some equations which basically say the field At
# is the integral of PiAt with respect to time. That is, we implement
# free evolution for At.

[1:1]       := D_CNT(At,t)
                 - AVGT(PiAt,t);

[2:2] := D_CNT(At,t)
                 - AVGT(PiAt,t);
# Apply quadratic interpolation to determine At at the second spatial
# point.
# [2:2] := <1>At[0]
#                 - (1/4)*( 3*(<1>At[-1]) + <1>At[1] );

# Include dissipation.
[3:Nr-2] := D_CNT(At,t)
                 - AVGTD( DISS(At,t), t)
                 - AVGT(PiAt,t);

[Nr-1:Nr-1] := D_CNT(At,t)
                 - AVGT(PiAt,t);

[Nr:Nr] := D_CNT(At,t)
                 - AVGT(PiAt,t);

}


evaluate residual Ar {

# Here we include some equations which basically say the field Ar
# is the integral of PiAr with respect to time. That is, we implement
# free evolution for Ar.

# Ar must be identically zero at r=0.
[1:1]       := <1>Ar[0];

[2:2] := D_CNT(Ar,t)
                 - AVGT(PiAr,t);
```

```
# Apply linear interpolation to determine Ar at the second spatial
# point.
# [2:2] := <1>Ar[0]
#                 - (1/2)*( <1>Ar[-1] + <1>Ar[1] );

# Include dissipation.
[3:Nr-2] := D_CNT(Ar,t)
                 - AVGTD( DISS(Ar,t), t)
                 - AVGT(PiAr,t);

[Nr-1:Nr-1] := D_CNT(Ar,t)
                 - AVGT(PiAr,t);

[Nr:Nr] := D_CNT(Ar,t)
                 - AVGT(PiAr,t);

}


residual rho {

    [1:Nr] := <1>rho[0]
                 - 2*e*( (<1>Phi1[0])*(<1>PiPhi2[0])
                 - (<1>Phi2[0])*(<1>PiPhi1[0]) )
                 + 2*(e^2)*(<1>At[0])*( (<1>Phi1[0])^2
                 + (<1>Phi2[0])^2 );

}



########################################################################
# Provide initializations for Phi1, Phi2, OmegaPhi1, OmegaPhi2,
# PiPhi1, PiPhi2, At, Ar, OmegaAt, OmegaAr, and rho.
#
# Time symmetric initial data is chosen for Phi1 and Phi2.
########################################################################

initialize <0>Phi1 {
[1:1] := 0;
[2:Nr-1] := amp1*exp(-(r-rcen1)^2/rwid1^2);
[Nr:Nr] := 0;
}

initialize <0>Phi2 {
[1:1] := 0;
[2:Nr-1] := amp2*exp(-(r-rcen2)^2/rwid2^2);
[Nr:Nr] := 0;
}

initialize <0>OmegaPhi1 {
[1:1] := 0;
```

```
# [2:Nr-1] := (-2*(r-rcen1)/rwid1^2)*amp1*exp(-(r-rcen1)^2/rwid1^2);
[2:Nr-1] := D_CNR(<0>Phi1[0],r);
[Nr:Nr] := 0;
}

initialize <0>OmegaPhi2 {
[1:1] := 0;
# [2:Nr-1] := (-2*(r-rcen2)/rwid2^2)*amp2*exp(-(r-rcen2)^2/rwid2^2);
[2:Nr-1] := D_CNR(<0>Phi2[0],r);
[Nr:Nr] := 0;
}

initialize <0>PiPhi1 {
[1:1] := 0;
[2:Nr-1] := 0;
[Nr:Nr] := 0;
}

initialize <0>PiPhi2 {
[1:1] := 0;
[2:Nr-1] := 0;
[Nr:Nr] := 0;
}

initialize <0>At {
[1:1] := 0;
[2:Nr-1] := amp1*exp(-(r-rcen1)^2/rwid1^2);
[Nr:Nr] := 0;
}

initialize <0>Ar {
[1:1] := 0;
[2:Nr-1] := amp2*exp(-(r-rcen2)^2/rwid2^2);
[Nr:Nr] := 0;
}

initialize <0>OmegaAt {
[1:1] := 0;
# [2:Nr-1] := (-2*(r-rcen1)/rwid1^2)*amp1*exp(-(r-rcen1)^2/rwid1^2);
[2:Nr-1] := D_CNR(<0>At[0],r);
[Nr:Nr] := 0;
}

initialize <0>OmegaAr {
[1:1] := 0;
# [2:Nr-1] := (-2*(r-rcen2)/rwid2^2)*amp2*exp(-(r-rcen2)^2/rwid2^2);
[2:Nr-1] := D_CNR(<0>Ar[0],r);
[Nr:Nr] := 0;
}


initialize <0>rho {
    [1:Nr]      := 2*e*((<0>Phi1[0])*(<0>PiPhi2[0])
                  - (<0>Phi2[0])*(<0>PiPhi1[0]))
```

```
                  - 2*(e^2)*(<0>At[0])*((<0>Phi1[0])^2 + (<0>Phi2[0])^2);
}


#######################################################################
# Instruct RNPL automatically initialize some fields.
#######################################################################
auto initialize Phi1, Phi2, OmegaPhi1, OmegaPhi2, PiPhi1, PiPhi2,
At, Ar, OmegaAt, OmegaAr, rho

#######################################################################
# Instruct RNPL to initialize other quantities via a different scheme.
#######################################################################

init_cnst.inc init_cnst INITIALIZE PiAt, PiAr HEADER
  PiAt[PiAtn,PiAt], PiAr[PiArn,PiAr],
  PiPhi1[PiPhi1n,PiPhi1], PiPhi2[PiPhi2n,PiPhi2],
  Phi1[Phi1n,Phi1], Phi2[Phi2n,Phi2],
  OmegaAt[OmegaAtn,OmegaAt], OmegaAr[OmegaArn,OmegaAr],
  At[Atn,At],
  Ar[Arn,Ar],
  rho[rhon,rho],
  e, r, rmin, rmax

init_Q.inc init_Q INITIALIZE Q HEADER
  Q, Phi1[Phi1n,Phi1], Phi2[Phi2n,Phi2],
  PiPhi1[PiPhi1n,PiPhi1], PiPhi2[PiPhi2n,PiPhi2],
  At[Atn,At],
  e, r, rmin, rmax

init_Ekg.inc init_Ekg INITIALIZE Ekg HEADER
  Ekg,
  PiPhi1[PiPhi1n,PiPhi1], PiPhi2[PiPhi2n,PiPhi2],
  OmegaPhi1[OmegaPhi1n,OmegaPhi1], OmegaPhi2[OmegaPhi2n,OmegaPhi2],
  Phi1[Phi1n,Phi1], Phi2[Phi2n,Phi2],
  m, r, rmin, rmax

init_Es.inc init_Es INITIALIZE Es HEADER
  Es,
  PiPhi1[PiPhi1n,PiPhi1], PiPhi2[PiPhi2n,PiPhi2],
  OmegaPhi1[OmegaPhi1n,OmegaPhi1], OmegaPhi2[OmegaPhi2n,OmegaPhi2],
  Phi1[Phi1n,Phi1], Phi2[Phi2n,Phi2],
  PiAt[PiAtn,PiAt], PiAr[PiArn,PiAr],
  OmegaAt[OmegaAtn,OmegaAt], OmegaAr[OmegaArn,OmegaAr],
  At[Atn,At],Ar[Arn,Ar],
  m, e, r, rmin, rmax
```

```
########################################################################
# Instruct RNPL to solve the equations iteratively and to automatically
# generate the update routines for some variables.
########################################################################
looper iterative

auto update Phi1, Phi2, OmegaPhi1, OmegaPhi2, PiPhi1, PiPhi2,
At, Ar, OmegaAt, OmegaAr, rho, PiAt, PiAr

########################################################################
# We provide the update routines for other quantities.
########################################################################

calc_Q.inc calc_Q UPDATES Q HEADER
  Q, Phi1[Phi1n,Phi1], Phi2[Phi2n,Phi2],
  PiPhi1[PiPhi1n,PiPhi1], PiPhi2[PiPhi2n,PiPhi2],
  At[Atn,At],
  e, r, rmin, rmax

calc_Ekg.inc calc_Ekg UPDATES Ekg HEADER
  Ekg,
  PiPhi1[PiPhi1n,PiPhi1], PiPhi2[PiPhi2n,PiPhi2],
  OmegaPhi1[OmegaPhi1n,OmegaPhi1], OmegaPhi2[OmegaPhi2n,OmegaPhi2],
  Phi1[Phi1n,Phi1], Phi2[Phi2n,Phi2],
  m, r, rmin, rmax

calc_Es.inc calc_Es UPDATES Es HEADER
  Es,
  PiPhi1[PiPhi1n,PiPhi1], PiPhi2[PiPhi2n,PiPhi2],
  OmegaPhi1[OmegaPhi1n,OmegaPhi1], OmegaPhi2[OmegaPhi2n,OmegaPhi2],
  Phi1[Phi1n,Phi1], Phi2[Phi2n,Phi2],
  PiAt[PiAtn,PiAt], PiAr[PiArn,PiAr],
  OmegaAt[OmegaAtn,OmegaAt], OmegaAr[OmegaArn,OmegaAr],
  At[Atn,At],Ar[Arn,Ar],
  m, e, r, rmin, rmax
```

B Initial Constraint Solver

Source file: `init_cnst.inc`

```
C=======================================================================
C     Routine to solve the initial constraint equation for the system
C     and return the values of PiAt and PiAr.
C
C     The initial constraint equation is
C
C     U''(r) + (2/r)*U'(r) = rho,
C
C     where rho is the charge density (as defined by variables Phi1,
C     Phi2, PiPhi1, PiPhi2, and At). Here, ' denotes differentiation
C     with respect to r.
C
C     The radial component of the electric field is then given by
C
C     Er(r) = - U'(r).
C
C     The value of PiAr is given by the electromagnetic stress energy
C     tensor. The equation we solve is
C
C     PiAr(r) = OmegaAt(r) - Er(r).
C
C     Furthermore, the value of PiAt is given by the Lorentz gauge
C     condition. Specifically, we solve
C
C     PiAt(r) = OmegaAr(r) + (2/r)*Ar(r)
C
C     This routine uses second-order finite difference techniques and
C     LAPACK tridiagonal solver DGTSV.
C
C     Borrows from bvp1d.f by Matt Choptuik, Department of Physics and
C     Astronomy, University of British Columbia.
C=======================================================================


      character*11 cdnm
      parameter       ( cdnm = 'init_cnst' )

      logical ltraceU
      parameter   ( ltraceU = .false. )

      logical ltraceEr
      parameter   ( ltraceEr = .false. )

      logical ltracePiAt
      parameter   ( ltracePiAt = .false. )

      logical ltracePiAr
      parameter   ( ltracePiAr = .false. )
```

```
C-----------------------------------------------------------------------
C     Maximum problem size (2**20 + 1).
C-----------------------------------------------------------------------
      integer         maxn
      parameter     ( maxn = 1 048 577 )
C-----------------------------------------------------------------------
C     Storage for U(r) and Er(r).
C-----------------------------------------------------------------------
      real*8          U(maxn),Er(maxn)
C-----------------------------------------------------------------------
C     Storage for main, upper and lower diagonals of tridiagonal system,
C     and right-hand-side vector for use with LAPACK routine DGTSV.
C-----------------------------------------------------------------------
      real*8          d(maxn),         du(maxn),
     &                dl(maxn),        rhs(maxn)
      integer         nrhs,            info
C-----------------------------------------------------------------------
C     Discretization level and size of system (# of discrete
C     unknowns).
C-----------------------------------------------------------------------
      integer         nr,                i
C-----------------------------------------------------------------------
C     Mesh spacing.
C-----------------------------------------------------------------------
      real*8          dr


C=======================================================================
C     First solve for U(r).
C=======================================================================

C-----------------------------------------------------------------------
C     Set nr equal to the number of grid points.
C-----------------------------------------------------------------------
      nr = g1_Nr
      if( nr .gt. maxn ) then
         write(0,*) 'Insufficient internal storage'
         stop
      end if
C-----------------------------------------------------------------------
C     Determine spacing of the finite-difference 'mesh'.
C
C     There is no need to set up the mesh itself, since RNPL does it
C     for us.
C-----------------------------------------------------------------------
      dr = (rmax - rmin) / (nr - 1)

C=======================================================================
C     Set up tridiagonal system to solve for U(r). Note that indexing on
C     lower diagonal is always (i-1) when implementing the i'th
C     equation.
C=======================================================================

C-----------------------------------------------------------------------
```

```
C     Left boundary, U(1).
C-----------------------------------------------------------------------
      d(1)       = -6.0d0*(dr**2)
      du(1)      = 6.0d0*(dr**2)
      rhs(1)     = rho(2)
C-----------------------------------------------------------------------
C     Interior. Use the second order FDA of the differential equation.
C-----------------------------------------------------------------------
      do i = 2 , nr - 1
         dl(i-1) = 1.0d0/(dr**2) - 1.0d0/(( r(i) )*( dr ))
         d(i)    = -2.0d0/(dr**2)
         du(i)   = 1.0d0/(dr**2) + 1.0d0/(( r(i) )*( dr ))
         rhs(i)  = rho(i)
      end do
C-----------------------------------------------------------------------
C     Right boundary, U(nr).
C-----------------------------------------------------------------------
      dl(nr-1)    = (2.0d0/(dr**2))
     &              * (( nr - 4.0d0 )/( nr - 2.0d0 ))
      d(nr)       = (2.0d0/(dr**2))
     &              * (( nr**2 - 4.0d0*nr + 1.0d0 )
     &              / (( nr - 2.0d0 )*( nr - 1.0d0 )))
      rhs(nr)     = rho(nr-1)

C=======================================================================
C     Solve tridiagonal system.
C=======================================================================

      nrhs = 1
      call dgtsv( dr, nrhs, dl, d, du, rhs, nr, info )
      if( info .eq. 0 ) then
C-----------------------------------------------------------------------
C        If solver successful, continue with the constraint solver. If
C        not, send message to standard error.
C
C        Tracing allows solution to be written to standard error.
C
C        For clarity, we write the values of rhs(i) to U(i).
C-----------------------------------------------------------------------
         do i = 1, nr
            U(i) = rhs(i)
         end do
         if ( ltraceU ) then
            do i = 1 , nr
               write(0,*) 'r(i) = ',r(i), 'rho(i) = ', rho(i)
               write(0,*) 'r(i) = ',r(i), 'U(i) = ', U(i)
            end do
         write(0,*) cdnm, ': U calculated.'
         else
            continue
         end if
      else
C-----------------------------------------------------------------------
C        Solver failed.
```

```
C-----------------------------------------------------------------------
            write(0,*) 'calc_cnst: dgtsv() failed, info = ', info
            stop
         end if


C=======================================================================
C     Now solve for Er(r).
C=======================================================================

C-----------------------------------------------------------------------
C     Left boundary, Er(1). Use forwards differencing.
C-----------------------------------------------------------------------
      Er(1) = -( -3.0d0*U(1) + 4.0d0*U(2) - U(3) )/(2.0d0*dr)
C-----------------------------------------------------------------------
C     Interior.
C-----------------------------------------------------------------------
      do i = 2 , nr - 1
         Er(i) = -( U(i+1) - U(i-1) )/(2.0d0*dr)
      end do
C-----------------------------------------------------------------------
C     Right boundary, Er(nr). Use backwards differencing.
C-----------------------------------------------------------------------
      Er(nr) = -( 3.0d0*U(nr) - 4.0d0*U(nr-1) + U(nr-2) )/(2.0d0*dr)

C-----------------------------------------------------------------------
C     Tracing allows solution to be written to standard error.
C-----------------------------------------------------------------------
      if ( ltraceEr ) then
         do i = 1 , nr
            write(0,*) 'r(i) = ',r(i), 'U(i) = ', U(i)
            write(0,*) 'r(i) = ',r(i), 'Er(i) = ', Er(i)
         end do
         write(0,*) cdnm, ': Er calculated.'
      else
         continue
      end if


C=======================================================================
C     Now solve for PiAr.
C=======================================================================

      do i = 1, nr
         PiAr(i) = OmegaAt(i) - Er(i)
      end do
C-----------------------------------------------------------------------
C     Tracing allows solution to be written to standard error.
C-----------------------------------------------------------------------
      if ( ltracePiAr ) then
         do i = 1, nr
            write(0,*) 'r(i) = ',r(i), 'OmegaAt(i) = ', OmegaAt(i)
            write(0,*) 'r(i) = ',r(i), 'Er(i) = ', Er(i)
            write(0,*) 'r(i) = ',r(i), 'PiAr(i) = ', PiAr(i)
```

```
         end do
         write(0,*) cdnm, ': PiAr initialized.'
      endif


C=========================================================================
C     Now solve for PiAt.
C=========================================================================

C-------------------------------------------------------------------------
C     Use L'hopital's rule for the r=0 boundary.
C-------------------------------------------------------------------------
      PiAt(1) = 3.0d0*OmegaAr(1)
C-------------------------------------------------------------------------
C     Calculate PiAt for all other values of r.
C-------------------------------------------------------------------------
      do i = 2, nr
         PiAt(i) = OmegaAr(i) + (2.0d0/(r(i)))*Ar(i)
      end do
C-------------------------------------------------------------------------
C     Tracing allows solution to be written to standard error.
C-------------------------------------------------------------------------
      if ( ltracePiAt ) then
         do i = 1, nr
            write(0,*) 'r(i) = ',r(i), 'OmegaAr(i) = ', OmegaAr(i)
            write(0,*) 'r(i) = ',r(i), 'Ar(i) = ', Ar(i)
            write(0,*) 'r(i) = ',r(i), 'PiAt(i) = ', PiAt(i)
         end do
         write(0,*) cdnm, ': PiAt initialized.'
      endif
```

C Representative Initial Data

Source file: `id2`

```
########################################################################
# Input parameters for EM
########################################################################

rmin := 0.0
rmax := 100.0
m := 1.0
e := 0.05
amp1 := 1.0
amp2 := 1.0
rcen1 := 50.0
rcen2 := 50.0
rwid1 := 3.0
rwid2 := 3.0
epsdis := 0.50


start_t := 0
s_start := 0
iter := 1000
epsiter := 1.0e-9
fout := 1
ser := 0
lambda := 0.2
output := *-*/1
in_file := "3Din2.sdf"
out_file := "3Dout2.sdf"
level := 2
tag := ""
Nr0 := 100

maxstepid := 1000
epsiterid := 1.0e-9

out2ser := 0
```

D Total System Energy Solver

Source file: calc_Es.inc

```
C=======================================================================
C     Routine to compute the system energy.
C=======================================================================

      character*11 cdnm
      parameter       ( cdnm = 'calc_Es' )

      logical ltrace
      parameter   ( ltrace = .false. )

      real*8 dr,pi4

      integer i,nr

C-----------------------------------------------------------------------
C     Initialize Es to 0.0d0.
C-----------------------------------------------------------------------
      Es_n(1) = 0.0d0

C-----------------------------------------------------------------------
C     Set Nr equal to the number of grid points.
C-----------------------------------------------------------------------
      nr = g1_Nr
C-----------------------------------------------------------------------
C     Determine grid spacing dr and pi4.
C-----------------------------------------------------------------------
      dr = (rmax - rmin)/(nr-1)
      pi4 = 16.0d0*atan(1.0d0)

      if ( ltrace ) then
         write(0,*) cdnm, ': nr=',nr
         write(0,*) cdnm, ': dr=',dr
      endif

C-----------------------------------------------------------------------
C     Main integration loop.
C-----------------------------------------------------------------------
      do i = 1 , nr-1
         Es_n(i+1) = Es_n(i) + pi4*(0.5d0)*(
     &               (  (PiPhi1(i))**2 + (PiPhi2(i))**2
     &                + (OmegaPhi1(i))**2 + (OmegaPhi2(i))**2
     &                + (m**2)*( (Phi1(i))**2 + (Phi2(i))**2 )
     &                + (2.0d0)*e*(At(i))*(
     &                     (Phi2(i))*(PiPhi1(i))
     &                   - (Phi1(i))*(PiPhi2(i)) )
     &                + (2.0d0)*e*(Ar(i))*(
     &                     (Phi2(i))*(OmegaPhi1(i))
     &                   - (Phi1(i))*(OmegaPhi2(i)) )
     &                + (e**2)*( (At(i))**2 + (Ar(i))**2 )
     &                  *( (Phi1(i))**2 + (Phi2(i))**2 )
```

```
&                 + (0.5d0)*( (PiAr(i))**2
&                    - (2.0d0)*( (PiAr(i))*(OmegaAt(i)) )
&                    + (OmegaAt(i))**2 )
&                ) *(r(i))**2
&              + (  (PiPhi1(i+1))**2 + (PiPhi2(i+1))**2
&                 + (OmegaPhi1(i+1))**2 + (OmegaPhi2(i+1))**2
&                 + (m**2)*( (Phi1(i+1))**2 + (Phi2(i+1))**2 )
&                 + (2.0d0)*e*(At(i+1))*(
&                     (Phi2(i+1))*(PiPhi1(i+1))
&                   - (Phi1(i+1))*(PiPhi2(i+1)) )
&                 + (2.0d0)*e*(Ar(i+1))*(
&                     (Phi2(i+1))*(OmegaPhi1(i+1))
&                   - (Phi1(i+1))*(OmegaPhi2(i+1)) )
&                 + (e**2)*( (At(i+1))**2 + (Ar(i+1))**2 )
&                   *( (Phi1(i+1))**2 + (Phi2(i+1))**2 )
&                 + (0.5d0)*( (PiAr(i+1))**2
&                    - (2.0d0)*( (PiAr(i+1))*(OmegaAt(i+1)) )
&                    + (OmegaAt(i+1))**2 )
&                ) *(r(i+1))**2
&                ) *dr
 if ( ltrace ) then
    write(0,*) cdnm, ': pi4 =',pi4
    write(0,*) cdnm, ': r(i) =',r(i)
    write(0,*) cdnm, ': Es(i)=',Es_n(i)
    write(0,*) cdnm, ': Es(i+1)=',Es_n(i+1)
 endif
 enddo

 if ( ltrace ) then
    write(0,*) cdnm, ': Es calculated'
 endif
```

E Klein Gordon Energy Solver

Source file: `calc_Ekg.inc`

```
C=====================================================================
C     Routine to compute the scalar field energy.
C=====================================================================

      character*11 cdnm
      parameter       ( cdnm = 'calc_Ekg' )

      logical ltrace
      parameter   ( ltrace = .false. )

      real*8 dr,pi4

      integer i,nr

C---------------------------------------------------------------------
C     Initialize Ekg to 0.0d0.
C---------------------------------------------------------------------
      Ekg_n(1) = 0.0d0

C---------------------------------------------------------------------
C     Set Nr equal to the number of grid points.
C---------------------------------------------------------------------
      nr = g1_Nr
C---------------------------------------------------------------------
C     Determine grid spacing dr and pi4.
C---------------------------------------------------------------------
      dr = (rmax - rmin)/(nr-1)
      pi4 = 16.0d0*atan(1.0d0)

      if ( ltrace ) then
         write(0,*) cdnm, ': nr=',nr
         write(0,*) cdnm, ': dr=',dr
      endif

C---------------------------------------------------------------------
C     Main integration loop.
C---------------------------------------------------------------------
      do i = 1 , nr-1
         Ekg_n(i+1) = Ekg_n(i) + pi4*(0.5d0)*(
     &               (  (PiPhi1(i))**2 + (PiPhi2(i))**2
     &                + (OmegaPhi1(i))**2 + (OmegaPhi2(i))**2
     &                + (m**2)*( (Phi1(i))**2 + (Phi2(i))**2 )
     &               ) *(r(i))**2
     &             + (  (PiPhi1(i+1))**2 + (PiPhi2(i+1))**2
     &                + (OmegaPhi1(i+1))**2 + (OmegaPhi2(i+1))**2
     &                + (m**2)*( (Phi1(i+1))**2 + (Phi2(i+1))**2 )
     &               ) *(r(i+1))**2
     &               ) *dr
      if ( ltrace ) then
         write(0,*) cdnm, ': pi4 =',pi4
```

```
   write(0,*) cdnm, ': r(i) =',r(i)
   write(0,*) cdnm, ': Ekg(i)=',Ekg_n(i)
   write(0,*) cdnm, ': Ekg(i+1)=',Ekg_n(i+1)
endif
enddo

if ( ltrace ) then
   write(0,*) cdnm, ': Ekg calculated'
endif
```

F Total Charge Solver

Source file: `calc_Q.inc`

```
C=======================================================================
C     Routine to compute the total charge given grid function rho, the
C     number of grid points, and the bounds of the grid.
C=======================================================================


      character*11 cdnm
      parameter       ( cdnm = 'calc_Q' )

      logical ltrace
      parameter   ( ltrace = .false. )

      real*8 dr,pi4

      integer i,nr


C-----------------------------------------------------------------------
C     Set nr equal to the number of grid points.
C-----------------------------------------------------------------------
      nr = g1_Nr
C-----------------------------------------------------------------------
C     Determine mesh spacing dr, and constant pi4=3.14159... .
C-----------------------------------------------------------------------
      dr = (rmax - rmin)/(nr-1)
      pi4 = 16.0d0*atan(1.0d0)

C=======================================================================
C     Integration.
C=======================================================================

C-----------------------------------------------------------------------
C     Initialize Q(1) to 0.0d0.
C-----------------------------------------------------------------------
      Q_n(1) = 0.0d0
C-----------------------------------------------------------------------
C     Integration loop.
C-----------------------------------------------------------------------
      do i = 1, nr-1
         Q_n(i+1) = Q_n(i) + pi4*(0.5d0)*(
     &           (
     &             (2.0d0)*e*(  (Phi1(i))*(PiPhi2(i))
     &                        - (Phi2(i))*(PiPhi1(i))
     &                       )
     &           - (2.0d0)*(e**2)*(At(i))*( (Phi1(i))**2
     &                                    + (Phi2(i))**2
     &                                    )
     &           ) *(r(i))**2
     &         + (
     &             (2.0d0)*e*(  (Phi1(i+1))*(PiPhi2(i+1))
```

```
&                          - (Phi2(i+1))*(PiPhi1(i+1))
&                         )
&            - (2.0d0)*(e**2)*(At(i+1))*( (Phi1(i+1))**2
&                                         + (Phi2(i+1))**2
&                                         )
&            ) *(r(i+1))**2
&            ) *dr
 if ( ltrace ) then
    write(0,*) cdnm, ': pi4 =',pi4
    write(0,*) cdnm, ': nr=',nr
    write(0,*) cdnm, ': dr=',dr
    write(0,*) cdnm, ': r(i) =',r(i)
    write(0,*) cdnm, ': Q(i)=',Q_n(i)
    write(0,*) cdnm, ': Q(i+1)=',Q_n(i+1)
 end if
 end do
 if ( ltrace ) then
    write(0,*) cdnm, ': Q calculated.'
 end if
```

References

[1] Marsa, R., *Radiative Problems in Black Hole Spacetimes* (Ph.D. thesis), (University of Texas at Austin, 1995).

[2] Hod, S., and Piran, T., *Critical Behaviour in Gravitational Collapse of a Charged Scalar Field*, (gr-qc/9606093 v3, LANL preprint archives, 1996).

[3] Liebling, S., *Nonlinear Field Dynamics in General Relativity: Black Hole Critical Phenomena and Topological Defects* (Ph.D. thesis), (University of Texas at Austin, 1998).

[4] Hawking, S., and Ellis, G., *The Large Scale Structure of Space-time*, (Cambridge University Press, New York, 1995).

[5] Ryder, L. H., *Quantum Field Theory*, (Cambridge University Press, New York, 1996).

[6] Choptuik, M. W., *Numerical Methods for General Relativity* (M.Sc. thesis), (University of British Columbia, 1982).

[7] Choptuik, M. W., *Numerical Techniques for Radiative General Relativity* (Ph.D. thesis), (University of British Columbia, 1986).

[8] Choptuik, M. W., *Physics 410 online course notes*, (http://laplace.physics.ubc.ca/People/matt/410/).

[9] York, J. W., and Piran, T., *The Initial Value Problem and Beyond*; from *Spacetime and Geometry*, editors Matzner, R., and Shepley, C., (University of Texas Press, Austin, 1982).