ad
An Implementation of the Berger-Oliger Mesh Refinement Algorithm for the Wave Equation in Spherical Symmetry

Matthew W. Choptuik
Center for Relativity
The University of Texas at Austin
Austin, TX, 78712-1081
matt@infeld.ph.utexas.edu

March 1994

1 Introduction

This document gives a brief overview of my FORTRAN 77 implementation of the Berger & Oliger algorithm [1] for hyperbolic problems in one space and one time dimension. It is strongly recommended that this documentation be read in conjunction with the Berger & Oliger paper. The code which is described here—ad—is a "stripped-down" version of a code which solves Einstein's equations coupled to scalar fields in spherical symmetry [2, 3, 4]. The reader should also consult the source code in ad/src as well as the control and README files in ad/run for more detailed information.

2 Getting the Source Code

Source code, documentation, sample control files and some utilities are available via anonymous ftp from feynman.ph.utexas.edu. It is assumed that the code will be installed on a "typical" scientific workstation (Sun, SGI, IBM RS6000, Dec Alpha, ...) running Unix, and that C and FORTRAN 77 compilers and a perl interpreter are available.

To retrieve ad, cd to the directory where you want ad installed, then enter:

```
ftp feynman.ph.utexas.edu
anonymous
<user name>@
```

Once the ftp connection has been established, type

```
bin
cd pub
get ad.tar.Z
quit
```

Now execute

```
zcat ad.tar.Z | tar xvf -
```

to reconstruct the ad directory hierarchy. The file ad/README will tell you how to proceed.

3 The Model Problem

`ad` solves the wave equation in spherical symmetry using second-order finite difference techniques and adaptive mesh refinement. The fundamental dynamical variable is a scalar field, $\phi(r,t)$, whose evolution is governed by

$$\frac{\partial^2 \phi}{\partial t^2} = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial \phi}{\partial r}\right) = 3\frac{\partial}{\partial r^3}\left(r^2 \frac{\partial \phi}{\partial r}\right) \tag{1}$$

I rewrite this equation in first-order (in time) form, by introducing auxiliary variables Φ and Π defined by:

$$\Phi \equiv \frac{\partial \phi}{\partial r} \qquad\qquad \Pi \equiv \frac{\partial \phi}{\partial t} \tag{2}$$

With these definitions, equation (1) may be rewritten as:

$$\frac{\partial \Phi}{\partial t} = \frac{\partial \Pi}{\partial r} \qquad\qquad \frac{\partial \Pi}{\partial t} = 3\frac{\partial}{\partial r^3}\left(r^2 \Phi\right) \tag{3}$$

The wave equation is solved on the finite domain $0 \le r \le R$, $0 \le t \le T$. Introducing a standard finite difference notation for grid functions (with a `FORTRAN` convention for the spatial grid index):

$$f_j^n \equiv f\left(r_j, t^n\right) \equiv f\left((j-1)\,\triangle r\,, n\,\triangle t\right) \tag{4}$$

where $\triangle r$ and $\triangle t$ are the *uniform* spatial (radial) and temporal mesh-spacings respectively, I use the following second-order finite difference approximations of (3):

$$\frac{\Phi_j^{n+1} - \Phi_j^{n-1}}{2\,\triangle t} = \frac{\Pi_{j+1}^n - \Pi_{j-1}^n}{2\,\triangle r} \tag{5}$$

$$\frac{\Pi_j^{n+1} - \Pi_j^{n-1}}{2\,\triangle t} = 3\,\frac{r_{j+1}^2\,\Phi_{j+1}^n - r_{j-1}^2\,\Phi_{j-1}^n}{r_{j+1}^3 - r_{j-1}^3} \tag{6}$$

Note that the spatial index, j, runs over the values $1, 2, \ldots$, `NR`, where $\mathtt{NR} \equiv R/\triangle r + 1$, and the temporal index, n, runs over the values $0, 1 \ldots$, `NT`, where $\mathtt{NT} \equiv T/\triangle t + 1$. (5) and (6) are trivially solved for Φ_j^{n+1} and Π_j^{n+1}, yielding the basic algebraic equations used to advance the solution from one time-step to the next:

$$\Phi_j^{n+1} := \Phi_j^{n-1} + \frac{\Delta t}{\Delta r}\left(\Pi_{j+1}^n - \Pi_{j-1}^n\right) \tag{7}$$

$$\Pi_j^{n+1} := \Pi_j^{n-1} + \frac{6\,\Delta t}{r_{j+1}^3 - r_{j-1}^3}\left(r_{j+1}^2\,\Phi_{j+1}^n - r_{j-1}^2\,\Phi_{j-1}^n\right) \tag{8}$$

(see `subroutine elfppi()` and `subroutine elfpii()` in `unigrid.f`). Equations (7–8) are used to update *interior* grid function values: i.e. those with spatial grid indexes $2 \le j \le$ `NR`. At the boundaries $j = 1$, $j =$ `NR`, the following additional conditions are used (see `subroutine elfpp()` and `subroutine elfpi()` in `unigrid.f`):

$$\Phi_1^{n+1} := 0 \tag{9}$$

$$\Pi_1^{n+1} := \frac{4\Pi_2^{n+1} - \Pi_3^{n+1}}{3} \tag{10}$$

$$\Phi^{n+1}_{\mathtt{NR}} := \left(\frac{3}{\triangle t}+\frac{3}{\triangle r}+\frac{2}{r_{\mathtt{NR}}}\right)^{-1}\left(\frac{4\Phi^{n}_{\mathtt{NR}}-\Phi^{n-1}_{\mathtt{NR}}}{\triangle t}+\frac{4\Phi^{n+1}_{\mathtt{NR}-1}-\Phi^{n+1}_{\mathtt{NR}-2}}{\triangle r}\right) \tag{11}$$

$$\Pi^{n}_{\mathtt{NR}} := \left(\frac{3}{\triangle t}+\frac{3}{\triangle r}+\frac{2}{r_{\mathtt{NR}}}\right)^{-1}\left(\frac{4\Pi^{n}_{\mathtt{NR}}-\Pi^{n-1}_{\mathtt{NR}}}{\triangle t}+\frac{4\Pi^{n+1}_{\mathtt{NR}-1}-\Pi^{n+1}_{\mathtt{NR}-2}}{\triangle r}\right) \tag{12}$$

Equations (9–10) are based on regularity of the scalar field ϕ at the origin:

$$\lim_{r\to 0} \phi(r,t) = \phi_0(t) + r^2\,\phi_2(t) + \cdots \tag{13}$$

while (11–12) are based on the (presumed) outgoing-only nature of solutions of (1) at large r

$$\lim_{r\to\infty} r\phi(r,t) = f(r-t) \tag{14}$$

which implies

$$\lim_{r\to\infty} \frac{\partial\Phi}{\partial t} + \frac{\partial\Phi}{\partial r} + \frac{\Phi}{r} = O\left(r^{-3}\right) \approx 0 \tag{15}$$

$$\lim_{r\to\infty} \frac{\partial\Pi}{\partial t} + \frac{\partial\Pi}{\partial r} + \frac{\Pi}{r} = 0 \tag{16}$$

Equations (11–12) follow from (15–16) by replacement of first derivatives with second-order *backwards* differences applied at $(r_{\mathtt{NR}}, t^{n+1})$.

In order to regularize (damp high frequency components) the numerical solution, `ad` applies a form of dissipation advocated by Kreiss & Oliger [5] for use with wave-equation discretizations such as (7–8). For example, following the computation of the advanced values Φ^{n+1}_j from (7), dissipation is added via:

$$\Phi^{n+1}_j := \Phi^{n+1}_j - \frac{\epsilon}{16}\left(\Phi^{n-1}_{j+2} - 4\Phi^{n-1}_{j+1} + 6\Phi^{n-1}_{j} - 4\Phi^{n-1}_{j-1} + \Phi^{n-1}_{j-2}\right) \tag{17}$$

where $\epsilon > 0$ is an adjustable parameter which must satisfy $\epsilon < 1$ for stability. A precisely analogous update is applied to Π^{n+1}_j. The value of ϵ is implemented as a run-time parameter, `EPSDIS`, which is read from the control file `ad.control`. (See `subroutine adddis()` in `unigrid.f` and calls to `adddis()` in `ad.f` and `ctres.f`).

`ad` also uses the following discretization of (2b) to maintain an approximation of the scalar field ϕ itself:

$$\Pi^n_j = \frac{\phi^{n+1}_j - \phi^{n-1}_j}{2\,\triangle t} \tag{18}$$

This yields algebraic equations for the update of the mesh quantities ϕ^{n+1}_j:

$$\phi^{n+1}_j := \phi^{n-1}_j + 2\,\triangle t\,\Pi^n_j \tag{19}$$

which are implemented in `subroutine elfph()` in `unigrid.f`.

Finally, chiefly for diagnostic purposes, `ad` computes approximations to (i) the radial energy density, dm/dr, (ii) the spatial integral, m (known as the *mass aspect* function) of dm/dr and (iii) m/r. The following discrete forms are used (see `subroutine clcmas()` in `unigrid.f`):

$$\frac{dm}{dr}^{n+1}_{\ j} := 2\pi r_j^2\left(\Phi^{n+1\,2}_j + \Pi^{n+1\,2}_j\right) \tag{20}$$

$$m_j^{n+1} := m_{j-1}^{n+1} + \frac{1}{2}\left(r_j - r_{j-1}\right)\left(\frac{dm}{dr}^{n+1}_{j} - \frac{dm}{dr}^{n+1}_{j-1}\right) \quad (21)$$

$$\frac{m}{r}^{n+1}_{j} := \frac{m_j^{n+1}}{r_j} \quad (22)$$

4 The Mesh Refinement Algorithm: Data Structures

The data structures used in `ad` are essentially those described in [1]. The program maintains a tree structure of nested, aligned, uniform grids (meshes) which are characterized by the level, `L`, of discretization, `L` $= 1, 2, \cdots,$ `MAXLEV` where `L` $= 1$ (`CRSLEV`) is the coarsest (base) level of discretization, `MAXLEV` is the maximum allowed level of refinement (defined implicitly via the control file `ad.refctl`) and where the mesh spacings on the various refinement levels satisfy:

$$\triangle r_{\mathrm{L}} = \mathtt{RHO}\ \triangle r_{\mathrm{L}+1} \quad (23)$$

$$\triangle t_{\mathrm{L}} = \mathtt{RHO}\ \triangle t_{\mathrm{L}+1} \quad (24)$$

$$\triangle t_{\mathrm{L}} = \mathtt{DTBYDR}\ \triangle r_{\mathrm{L}} \quad (25)$$

Here, `RHO` is the user-specified (`ad.control`), integer-valued mesh-refinement ratio and `DTBYDR` is the user-specified (`ad.control`) real-valued *Courant number* which must be strictly less than 1.0 for stability.

The tree structure of grids is implemented using various `FORTRAN` arrays which are defined and documented in `gridst.inc`. At any given time, any grid in the structure is identified and manipulated using a unique integer-valued tag, `G`, and most of the grid manipulation routines (see `grid.f`) take one or more such tags as arguments.

One aspect of `ad` which deserves some explanation is my implementation of a "pointer/aliasing" mechanism for accessing a specific grid-function defined on a given grid (see also `ad/src/gfin` and the generated file `fcnst.inc`). This mechanism was designed to (i) maximize flexibility with respect to the introduction of new grid functions (ii) minimize the amount of source code needed to introduce a new grid function (iii) keep the source code as readable as possible. It should be noted that the utility of this approach is perhaps not as evident for `ad` as it is for the general relativistic code, where something in excess of 60 distinct grid functions are defined.

Each grid has a fixed number, `NFCN` (see `fcnst.inc`), of grid-functions defined on it. The number of grid points on a given grid, `G`, is `GNR(G)`, so a total of `NFCN` $\times$ `GNR(G)` words of storage are required for the grid. This storage is always allocated as a contiguous portion of the primary storage array `Q(SIZMAX)` (`fcnst.inc`) using the memory management routines in `mm.f`. The pointer to this block is maintained in `GPMLOC(G)` (`gridst.inc`). In addition, `NFCN` subpointers to length-`GNR(G)` blocks of storage for the individual grid functions are stored in the array section `GPFCN(G,1:NFCN)` (`gridst.inc`). Also defined in `fcnst.inc` are integer-valued "pointers" or "aliases": `PPN, PPNP1, PIN,` $\cdots$, `W3, W4` which are (i) placed in a `COMMON` block to guarantee contiguity of physical storage, and (ii) `EQUIVALENCEd` to an array `PTRS(NFCN)`. The invocation

```
CALL LDPTR1(G)
```

(see `grid.f`), copies the array slice `GPFCN(G,1:NFCN)` to `PTRS(1:NFCN)`. This has the effect of loading the aliases `PPN, PPNP1, PIN,` $\cdots$, `W3, W4` with appropriate indexes into the primary storage array `Q`. The routine also sets the values of `NR`, `DR` and `DT` (globally accessible via `gridst.inc`) to the corresponding grid-`G` values. Thus, for example, the code

```
CALL LDPTR1(G)
CALL DVLS(Q(ZERO),0.0D0,NR)
CALL DVVM(Q(PPN),Q(PIN),Q(W1),NR)
```

sets the `ZERO` function on `G` to 0.0's, and multiplies (pointwise) the functions `PPN` and `PIN` and places the result in `W1`.

To facilitate operations involving *two* distinct grids, a secondary set of pointers/aliases—`PPN2, PPNP12, PIN2, ⋯, W32, W42`—is defined and `EQUIVALENCEd` to `PTRS2(NFCN)`. The call

```
CALL LDPTR2(G1,G2)
```

(see `grid.f`) associates the primary pointers with grid `G1`'s storage and the secondary pointers with grid `G2`'s functions. Thus, assuming that grids `G1` and `G2` have the same number of grid points (`GNR(G1)` = `GNR(G2)`), the following code copies the `MASS` grid function from grid `G1` to `G2`:

```
CALL LDPTR2(G1,G2)
CALL DVCOPY(Q(MASS),Q(MASS2),NR)
```

In order to expedite the process of defining new grid functions, as well as to minimize errors in the construction of the alias layouts, the file `fcnst.inc` is "compiled" (using the `perl` script `ad/bin/makegf`) from a higher-level specification contained in `ad/src/gfin`. The reader is referred to that file for a more complete description of the syntax/semantics of this utility. Here, for the purposes of illustration, I note that the interpretable portion of the file consists of statements such as:

```
gf(PP,         "d(phi)/dr",    2,
   '((INT  1) (OUTG 0) (OUTL 1) (INJ  1) (TRAN 1) (EVLV 1) (DEF 0))');
```

and

```
gf(MASS,       "mass",          1,
   '((INT  1) (OUTG 0) (OUTL 0) (INJ  1) (TRAN 1) (EVLV 0) (DEF 0))');
```

This results in the definition of aliases `PPN` and `PPNP1` in the first instance—where the third argument to `gf()` is 2, and the alias `MASS` in the second—where `gf()`'s third argument is 1. (in addition, the secondary aliases `PPN2`, `PPNM12`, `MASS2` are also defined). The idea here is that some functions need to have storage for *two* time levels of data (`PPN` $\sim \Phi_j^n$, `PPNP1` $\sim \Phi_j^{n+1}$ or Φ_j^{n-1}), while others, such as m (`MASS`) require only *one* time level. (There is also a provision for functions which require *four* time levels of storage, but this capability is not used by `ad`).

It should also be noted that this pointer mechanism provides an efficient method of effecting a time-step advance on grid `G`: `N` and `NP1` pointer values are simply swapped, then up-loaded into `GPFCN(G,1:NFCN)` (see `subroutine swptr1()` in `grid.f`).

5 The Mesh Refinement Algorithm: Program Flow

The following is a schematic overview of the overall flow of `ad`. Most lines in this description correspond to a single subroutine call or function invocation in the actual source code. Refer to the source code (as well as [1]) for more details.

```
                                                              Source File
ad [MAIN]                                                     ad.f
  driver_ad(ctres0)                                           ad.f
    Initialize grid structure
    gsinit()                                                  grid.f
    Allocate coarsest (level 1) grid
    galloc(crslev,glrmin,glrmax,timeid)                       grid.f
    Determine initial data on coarsest grid
    idexact(lhead(crslev))                                    ad.f
      gfinit(g)                                               ad.f
        ccf                                                   unigrid.f
      idweq(g)                                                ad.f
        clcid_full                                            unigrid.f
        clcmas                                                unigrid.f
    Determine initial refinement structure
    WHILE truncation_error(finlev) > TREMAX DO
      Need another level of discretization: invoke regridding procedure
      regrid(ctres0)                                          regrid.f
        First stage of regridding procedure: trunc. err. estimation, clustering, buffering and merging
        rgrid1                                                regrid.f
          Generate truncation error estimate using Richardson procedure
          ctrest [ctres0]                                     ctres.f
            Make two copies of the grid: one for the coarse evolution, one for the fine
            gfine = gcopy(g,1)                                grid.f
            gcoars = gcopy(g,1)                               grid.f
            ldptr2(g,gcoars)                                  grid.f
            Evolve data backwards in time to produce retarded values on coarse mesh
            -elfppi(g) -> -gcoars                             unigrid.f
            -elfpii(g) -> -gcoars                             unigrid.f
            Take a step on the coarse mesh ...
            crstep(gcoars)                                    ctres.f
              elfppi(gcoars)                                  unigrid.f
              elfpii(gcoars)                                  unigrid.f
              swptr1(gcoars)                                  grid.f
            ldptr1(gfine)                                     grid.f
            ... and two steps on the fine mesh
            DO 1 .. 2
              elfppi(gfine)                                   unigrid.f
              elfpii(gfine)                                   unigrid.f
              adddis(gfine)                                   unigrid.f
              swptr1(gfine)                                   grid.f
            END DO
```

```
        Compute trunc. err. estimate from relativized point-wise subtraction of coarse/fine values
        ldptr2(gcoars,gfine)                          grid.f
        hctr2(ptrs1,ptrs2)                            ctres.f
        dvglmf etc.(q(estre2))                        utility.f
      Form clusters, add buffer regions and merge clusters
      rgflg1                                          regrid.f
      rgclfilter                                      regrid.f
      rgflg2                                          regrid.f
      rgexcln                                         regrid.f
      rcenbl                                          rc.f
    Second stage of regridding procedure: construct grid hierarchy from clusters
    rgrid2                                            regrid.f
      galloc(l,rmin,rmax,t)                           grid.f
      rginterp(g,0)                                   regrid.f
      Function values specified analytically at initial time
      idexact(g)                                      ad.f
END WHILE
Initial data determined ... integrate system for requested number of coarse steps
cycle(nt1,ctres0)                                     ad.f
  Regrid if it is time to do so
  rgrid(ctres0)                                       regrid.f
    rgrid1                                            regrid.f
      <AS ABOVE>
    rgrid2                                            regrid.f
      galloc(l,rmin,rmax,t)                           regrid.f
      Interpolate function values from parental grids
      rginterp(g,0)                                   regrid.f
      Transfer function values from old grids at same level
      rgtran(g,gp)                                    grid.f
      Free grids from previous hierarchy
      gdealloc(g)                                     grid.f
  Advance level-l difference equations
  tstep(l)                                            ad.f
    tstep1(l,evlv0)                                   ad.f
      Advance grid-g difference equations
      gstep1(g,evlv0)                                 ad.f
        swptr2(g)                                     grid.f
        Interpolate boundary values from parent grids
        setbv2(g)                                     ad.f
        fcevlv(g) [evlv0]                             ad.f
          evlv_lf0(g)                                 ad.f
            Difference equations are encoded in the elf... routines
            elfpp                                     unigrid.f
              elfppi                                  unigrid.f
            elfpi                                     unigrid.f
              elfpii                                  unigrid.f
            adddis                                    unigrid.f
```

Swap bi-level (`N`, `NP1`) pointers to effect time step advance	
`swptr1(g)`	`grid.f`
`gmisc(g)`	`ad.f`
`clcmas(g)`	`unigrid.f`
Interrupt-handling: interactive IO control	
`vsint`	`io.f`
Inject grid functions into parents	
`linject(l)`	`grid.f`
Execution trace and file output	
`trcout(l)`	`io.f`
`output(l)`	`io.f`

6 Running `ad`; Control Files

6.1 Input

`ad` should be invoked in `ad/run` or some similarly prepared directory. In particular, `ad` reads three `ASCII` control files:

```
ad.control
ad.loctl
ad.refctl
```

Explanation of the contents/use of these files is given in `ad/run/README`.

Since `ad` is intended purely for demonstration purposes, the initial data is currently "hardwired" (see `subroutine clcid_full()` in `unigrid.f`) to produce a single, initially imploding, Gaussian pulse in ϕ which hits the origin at $t \approx 50$, the explodes (becomes outgoing) and leaves the computational domain at $t \approx 160$

The following is a brief description of some of the key parameters the reader may wish to experiment with:

- `TREMAX` (`ad.control`)
  Truncation error criteria. This is the value of the (relativized) local truncation error which `ad` attempts to match by creating and destroying local refinements. The original value is `5.00e-03`: numerically larger values will result in less regridding, smaller values will induce more refinement.

- `RHO` (`ad.control`)
  Mesh refinement ratio—integer ratio of successive mesh spacings. Original value is `5`. Smallest possible value of `2` will result in many levels of refinement being used, but regridding operations will add significantly more "noise" to numerical solution.

- `ad.refctl`
  The number of (sequentially numbered) lines in this file controls the maximum level of refinement (`MAXLEV`) used in a run. (see `ad/run/README`).

6.2 Output

While it is running, ad writes various diagnostics to standard output. Upon termination it also produces a summary of cluster statistics (from regridding operations), the final grid and segment structures, the refinement control structure, and a map of the free block structure (memory management) in the file

```
ad.dump
```

ad can also be configured to output any grid function at arbitrary time steps and on arbitrary levels of refinement (see ad/run/README and ad/src/gfin) for subsequent perusal and/or analysis. Output is to FORTRAN UNFORMATTED files with extension .segdat. A .segdat file has a record structure defined implicitly by the following code which scans foo.segdat:

```
c
c         Maximum number of functions anticipated to be stored in a
c         given .segdat file ...
c
          INTEGER       MAX_NFCN
          PARAMETER   ( MAX_NFCN = 100 )
c
c         Maximum number of radial coordinates anticipated for any
c         given function profile ...
c
          INTEGER       MAX_NR
          PARAMETER   ( MAX_NR = 10 000 )
c
c         Storage for print names of functions, actual number of fcns ...
c
          CHARACTER*16  FNAME(MAX_NFCN)
          INTEGER       NFCN,           IFCN
c
c         Storage for radial coordinates, function profiles ...
c
          REAL*8        R(MAX_NR),      F(MAX_NR)
c
c         Output time, output index, number of radial coordinates,
c         local variable for implied-DO loops ...
c
          REAL*8        T
          INTEGER       IT,             NR,          J
c
c         Logical unit for reading .segdat file ...
c
          INTEGER       USEG
          PARAMETER   ( USEG = 10 )

          OPEN(USEG,FILE='foo.segdat',FORM='UNFORMATTED',STATUS='OLD',
     *         ERR=900)
```

```
c
c          Get number of functions stored, function print names ...
c
           READ(USEG,ERR=910,END=910) NFCN
           READ(USEG,ERR=910,END=910) ( FNAME(IFCN) , IFCN = 1 , NFCN )
c
c          Total number of time steps of data stored is not stored
c          explicitly: application program responsible for detecting
c          end of file ...
c
           IT = 0
 100       CONTINUE
c
c             Read time, number of radial coordinates stored at this time ...
c
              READ(USEG,ERR=910,END=200) T,      NR
c
c             Read radial coordinates ...
c
              READ(USEG,ERR=910,END=910) ( R(J) , J =  1 , NR )
c
c             Read function values ...
c
              DO IFCN = 1 , NFCN
                 READ(USEG,ERR=910,END=910) ( F(J) , J = 1 , NR )
              END DO
           GO TO 100
c
c          Normal exit ...
c
 200       CONTINUE
           WRITE(*,*) 'foo.segdat contains ', NFCN, ' functions and ',
     *                NT, ' distinct time steps of data'
           STOP
c
c          Error exits ...
c
 900       CONTINUE
              WRITE(*,*) 'Error opening foo.segdat'
           STOP

 910       CONTINUE
              WRITE(*,*) 'Error/unexpected end-of-file reading foo.segdat'
           STOP
```

`ad` produces segdat files which contain a single function each (`NFCN = 1`) and which have names of the form

```
<tag><compressed print name><level>.segdat
```

where `<tag> := abs(LSTATE)` (see `ad.control`), `<compressed print name>` is a character string formed by removing special characters from the print name defined for the function in `ad/src/gfin` and `<level>` is the discretization level. It should be noted that output at level `L` at a given time will contain grid unknowns from level `L` and *all finer levels*. The selection of which grid functions are to be output is controlled either through `ad/src/gfin` or with the `lfon` and `lfoff` utilities described below. Selection of which specific time steps are to be output (on a per-level basis) is controlled using index vectors defined in `ad/run/ad.loctl` and documented in `ad/run/README`.

7 Utilities

7.1 `segdump`

Information (function profiles) from `.segdat` files can be extracted using the utility `ad/bin/segdump` whose usage is:

```
segdump <segdat tag> [<fcn index> <time index vector> <format>]
```

where `<segdat tag>` is the part of the `.segdat` file name *before* the extension, `<fcn index>` is the index of the function to be extracted (always `1` for the `.segdat` files produced by `ad`), `<time index vector>` is the index vector (see `ad/run/README` for discussion of index vectors) of times steps (as stored in the `.segdat` file) to be extracted, and `<format>` is an optional `FORMAT` specification. If three or more of `segdump`'s arguments are specified, output is to standard output, otherwise the user is prompted for required arguments and output is directed to a file `<segdat tag>.listing`. If `<time index vector>` specifies a single time, output is two column: `R(J)  F(J)`, otherwise output is three column `R(J)  F(J)  T`. Typical invocations are:

```
segdump 1pi1 1 1
```

which lists the 5th profile from `1pi1.segdat` on standard output

```
segdump 1dmassdr1 1 '1-*/2'
```

which lists every second profile from `1dmassdr1.segdat` on standard output.

7.2 `lfn`, `lfon` and `lfoff`

These scripts are used to enable/disable `.segdat` output of selected grid functions (see `ad/src/gfin` for an alternate mechanism). They must be invoked in the `ad/src` directory. `lfn` produces a numbered list of all grid functions defined in the function structure `fcnst.inc`:

```
Unix% lfn
1       d(phi)/dr
2       d(phi)/dr
3       Pi
4       Pi
5       phi
6       phi
```

```
7       mass
8       mass/r
9       d(mass)/dr
10      trunc. err. est.
11      r
12      r**2
13      r**3
14      zero
15      work
16      work
17      work
18      work
```

`lfon` (with no arguments) produces a numbered list of all grid functions for which `.segdat` output is enabled:

```
Unix% lfon
1       d(phi)/dr
2       d(phi)/dr
3       Pi
4       Pi
9       d(mass)/dr
```

`lfon <fcn #> <fcn #>` $\cdots$ enables `.segdat` output for the specified functions (identified by number), and then outputs an updated list:

```
Unix% lfon 8
1       d(phi)/dr
2       d(phi)/dr
3       Pi
4       Pi
8       mass/r
9       d(mass)/dr
```

Note: if a function which has two time-levels of storage associated with it (such as `d(phi)/dr` or `Pi`) is to be enabled, *both* of the function indexes should be specified in the `lfon` command.

`lfoff` (no arguments) disables *all* `.segdat` output. `lfoff <fcn #> <fcn #>` $\cdots$ disables selected functions:

```
Unix% lfoff 1 2 8
3       Pi
4       Pi
9       d(mass)/dr
```

Use of these utilities may change the source code for `ad`*: thus the user should be careful to recompile* `ad` *(*`make ad`*) after their invocation.*

8 Acknowledgements

This work was supported by the Natural Sciences and Engineering Council of Canada, by NSF PHY-8603284 to S.A. Teukolsky, NSF PHY-8806567 and PHY-9310083 to R.A. Matzner, Texas Advanced Research Project TARP-085 to R.A. Matzner, a Cray Research Grant to R.A. Matzner, and ASC9318152 (ARPA supplemented) to the Binary Black Holes Grand Challenge Alliance

References

[1] M. J. Berger and J. Oliger, *J. Comp. Phys*, **53**, 484 (1984)

[2] M. W. Choptuik, in **Frontiers in Numerical Relativity**, Evans, Finn & Hobill, eds., 206–221 (1989)

[3] M. W. Choptuik, in **Approaches to Numerical Relativity**, R. d'Inverno, ed., 202–222 (1992)

[4] M. W. Choptuik, *Phys. Rev. Lett.,* **70**, 9–12 (1993)

[5] H.-O. Kreiss and J. Oliger, **Methods for the Approximate Solution of Time Dependent Problems**, GARP Publications Series No. 10, (1973)