Code `OllinAxis-BiB`
User's manual

Miguel Alcubierre
Instituto de Ciencias Nucleares, UNAM
malcubi@nucleares.unam.mx

June, 2022

Contents

1 Introduction **3**

2 Downloading the code **3**

3 Directory structure **4**
3.1 Compiling 5
3.2 Running 6

4 Parameter files **6**

5 Output files **7**

6 Numerical grid **8**
6.1 Grid structure and staggering of the axis 8
6.2 Box-in-box grid refinement 9

7 Spacetime and evolution equations **10**
7.1 Spacetime metric 10
7.2 Evolution equations 11

8 Matter **13**

9 Slicing conditions **13**

10 Shift conditions **13**

11 Initial data **13**

12 Numerical methods **13**
12.1 Time integration 13

13 List of main code parameters **13**

14 Editing the code **13**

15 Ollingraph **13**

1 Introduction

This program solves the Einstein evolution equations in axial symmetry using a curvilinear version of the BSSN formulation for the 3+1 evolution equations, with different types of matter and different gauge conditions.

The main difference of this version of the code with respect to previous ones is the fact that it uses box-in-box mesh refinement (hence the BiB part of the name), and is parallelized with MPI. Many parts of this code are based on a previous version written by myself and José Manuel Torres. An even older version was written by Milton Ruiz.

Note: This manual is TERRIBLY INCOMPLETE! I haven't had the time to do it. Still, it should give you a good idea of the basics. I will be adding a little bit more every now and again.

2 Downloading the code

If you are reading this it means you probably already downloaded the code. But if for some reason you need to download it here is how.

The easiest way to obtain the code is to download it from GitHub:

```
git clone https://github.com/malcubi/OllinAxis-BiB
```

Once you have the code, you can get updated versions by just doing "git pull" inside the code main directory.

3 Directory structure

The main directory for the code is `OllinAxis-BiB`. There are several sub-directories inside this main directory:

`CVS`	Contains information about the CVS root and server (see Sec. 14).
`doc`	Contains the tex and pdf files for this user's manual.
`exe`	This directory is created at compile time and contains the executable file. It also contains a copy of the parameter files found in directory `par` (see below).
`fakempi`	Contains fake MPI routines so that the compiler won't complain if MPI is not installed.
`gnuplot`	Contains a couple of simple gnuplot macros for visualization.
`objs`	This directory is created at compile time and contains all the object and module files.
`ollingraph`	Contains the visualization packages "ollingraph" and "ollingraph2D" for convenient "quick and dirty" visualization (see Section 15 below).
`par`	Contains examples of parameter files (see Section 4 below).
`prl`	Contains perl scripts used at compile time to create the subroutines that manage parameters and arrays.
`src`	Contains the source files for all the code routines.

The directory `src` is itself divided into a series of sub-directories in order to better classify the different routines. These sub-directories are:

`auto`	Contains `FORTRAN` files that are automatically generated at compile time by the perl scripts. These files should not be edited manually!
`base`	Contains the routines that control the basic execution of the code, including the parameter and array declarations, the parameter parser, the output routines, the main evolution controllers, and generic routines for calculating derivatives, dissipation, etc. The code in fact starts execution at the routine `main.f90` contained in this directory.

`elliptic`	Contains routines for solving elliptic equations for initial data and/or maximal slicing for example.
`geometry`	Contains routines related to initial data, evolution and analysis of the spacetime geometric variables, including sources, gauge conditions, constraints, horizon finders, etc.
`matter`	Contains routines related to the initial data, evolution and analysis of the different matter models, including a generic routine for calculating the basic matter variables, and routines for evolving scalar fields, electric fields, fluids, etc.

The code is written in `FORTRAN 90` and is parallelized with `MPI` (Message Passing Interface). All subroutines are in separate files inside the directory `src` and its sub-directories.

3.1 Compiling

To compile just move inside the `OllinAxis-BiB` directory and type:

```
make
```

This will first run some perl scripts that create a series of automatically generated `FORTRAN` files that will be placed inside the directory `src/auto`. It will then compile all the `FORTRAN` routines that it can find inside any of the sub-directories of `src` (it will attempt to compile any file with the extension `.f90`).

The resulting object files and `FORTRAN` module files will be placed inside the sub-directory `objs`. The Makefile will then create a directory `exe` and will place in it the final executable file called `ollinaxis`. It will also copy to this directory all the sample parameter files inside the sub-directory `par`, and the visualization package `ollingraph`.

Notice that at this time the Makefile can use the compilers `g95`, `gfortran`, or the Intel compilers `ifc` and `ifort`, and it will automatically check if they are installed. If you have a different compiler then the Makefile will have to be modified (hopefully it won't be very difficult). The code will also attempt to find an `MPI` installation (it looks for the command `mpif90`), and if it does not find it it will use the fake routines inside the directory `fakempi`.

The Makefile has several other useful targets that can be listed by typing: `make help`.

3.2 Running

To run the code move into the directory `exe` and type:

```
ollinaxis name.par
```

Where `name.par` is the name of your parameter file (more on parameter files below). The code will then read data from the parameter file silently and hopefully start producing some output to the screen. The code will also create an output directory and will write the data files to that directory.

For parallel runs using `MPI` one must use instead the command:

```
mpirun -np N ollinaxis name.par
```

where `N` should be an integer number that specifies the number of processors to be used.

4 Parameter files

At run time the code reads the parameter values from a parameter file (parfile), with a name of the form `name.par`, that must be specified in the command line after the executable:

```
ollinaxis name.par
```

The data in this parameter file can be given in any order, using the format:

```
parameter = value
```

Comments (anything after a `#`) and blank lines are ignored. Only one parameter is allowed per line, and only one value is allowed per parameter, with the exception of the parameters `outvars0D`, `outvars1D` and `outvars2D` that control which arrays get output and take lists of arrays as values, for example:

```
outvars0D = alpha,A,B,C,H
```

There is in fact one other parameter that can also take multiple values as input, it is the parameter `mattertype` that can accept several types of matter at once (see Section 8 below).

Parameters that do not appear in the parfile get the default values given in the file `src/base/param.f90`. Examples of parameter files can be found in the subdirectory `par`.

IMPORTANT: Even though `FORTRAN` does not distinguish between upper and lower case in variable names, the names of parameters are handled as strings by the parameter parser, so lower and upper case are in fact different. The name of parameters in the parameter file should then be identical to the way in which they appear in the file `param.f90`.

5 Output files

At run time, the codes creates an output directory whose name should be given in the parameter file. It then produces a series of output files with the data from the run. There are so called `0D` files (with extension `*.tl`), `1D` files (with extensions `*.rl`, `*.zl` and `*.dl`), and `2D` files (with extension `*.2D`).

The `0D` files refer to scalar quantities obtained from the spatial arrays as functions of time. These scalar quantities include the maximum (`max`), the minimum (`min`), and three different norms of the spatial arrays: maximum absolute value (`nm1`), root mean square (`nm2`), and total variation (`var`). The value of different variables at the origin is also output.

The `1D` files contain the complete arrays along the coordinate axis and diagonals at different times, while the `2D` files output the full arrays. Beware: If you do output very often these files can become quite large!

Since we can have several grid refinement levels, and grid boxes, the file names are appended with a number corresponding to the specific box and level (all grid levels have output). For example:

`alphab0l0.rl`: Level 0 (coarsest grid)
`alphab0l1.rl`: Level 1
...

All files are written in ASCII, and using a format adapted to XGRAPH (but other graphic

packages should be able to read them).

Output is controlled by the following parameters:

`directory`	Name of directory for output.
`Ninfo`	How often do we output information to screen?
`Noutput0D`	How often do we do 0D output?
`Noutput1D`	How often do we do 1D output?
`Noutput2D`	How often do we do 2D output?
`outvars0D`	Arrays that need 0D output (a list separated by commas).
`outvars1D`	Arrays that need 1D output (a list separated by commas).
`outvars2D`	Arrays that need 2D output (a list separated by commas).

6 Numerical grid

6.1 Grid structure and staggering of the axis

The code works in cylindrical coordinates (r, z), where r represents distance to the axis of symmetry (not distance to the origin!), and z the height above or below the "equator". Grid functions are represented by $f_(i, j)$, with the index i representing the r position, and the index j the z position.

In order to avoid having divisions by zero in some terms, we stagger the symmetry axis. This means that there is no grid point at $r = 0$. Instead, the first grid point (grid point 0) is located at $r = -dr/2$ (with dr the grid spacing), the second one (grid point 1) at $r = +dr/2$, and so on. This also makes it easy to apply symmetry boundary conditions, since for an even grid function $f_{(i,j)}$ we can simply take $f_{0,j} = +f_{1,j}$, and for an odd function $f_{0,j} = -f_{1,j}$.

Grid points to the left of symmetry axis are known as "ghost points", and the code usually adds more than one in order to be able to use higher order differencing stencils. The positions of the grid points along the r direction then correspond to $r_i = (i - 1/2) * dr$, where i runs

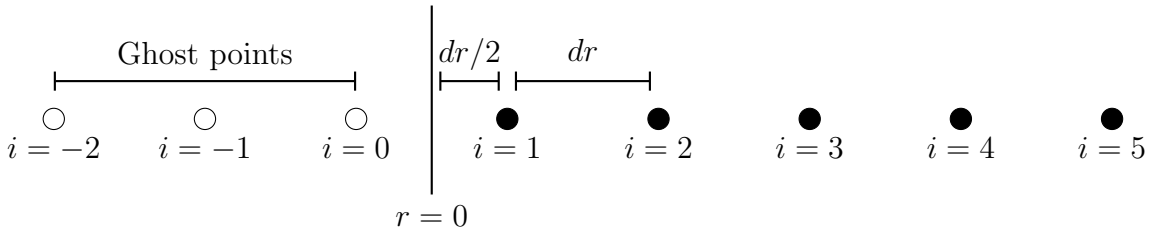

Figure 1: Basic grid structure close to the axis of symmetry $r = 0$, showing how the axis is staggered. We show the case of 3 ghost points to the left of the origin (for fourth order spatial differencing).

from $(1 - g)$ to some maximum number `Nr`, and where g is the number of ghost points given by the parameter `ghost`. In fact, for second order spatial differencing the code takes `ghost=2`, while for fourth order differencing it takes `ghost=3`, see Figure 1 (the reason for this has to do with dissipation, which typically needs one extra ghost point that would normally be required for a given differencing order, see below). Notice that the parameter `ghost` is determined by the code at run time and should NOT be fixed in the parameter file.

The grid for the coordinate z, on the other hand, has two different structures depending on whether we have or not equatorial symmetry. If the logical parameter `eqsym` is set to `.false.`, that is if there is no equatorial symmetry, then the grid uses `Nz` points along the z direction starting from $-Nz/2$ to $+Nz/2$. Notice that of `Nz` is even then there will be no point at $z = 0$ (and the equator will be staggered9, while if `Nz` is odd there will be a point with $z = 0$.
On the other hand, if the logical parameter `eqsym` is set to `.true.`, then the equator is always staggered and symmetry conditions are applied in the z direction in the same way as it is done with the axis if symmetry for the coordinate r. Having equatorial symmetry then reduces the use of computational by half.

6.2 Box-in-box grid refinement

The also code uses "box-in-box" type grid refinement. That is, it has refinement levels with higher resolution ...

7 Spacetime and evolution equations

7.1 Spacetime metric

As already mentioned, the code uses cylindrical coordinates (r, z) in space. Notice that what the code calls r here represents distance to the symmetry axis and not distance to the origin. The distance to the origin in the code is in fact called $rr := (r^2 + z^2)^{1/2}$.

The spatial metric is written in the following way:

$$\begin{aligned} dl^2 = e^{4\phi(r,z,t)} \big[& A(r,z,t)dr^2 + B(r,z,t)dz^2 + r^2 H(r,z,t)d\varphi^2 \\ & + 2\, r \left(C(r,z,t)drdz + r^2 C_1(r,z,t)drd\varphi + rC_2(r,z,t)dzd\varphi \right) \big] \ , \end{aligned} \tag{7.1}$$

with ϕ the conformal factor. The powers of r in some of the terms come from the behaviour of the different metric components close to the axis. Factorizong those powers of r exoplicitely allows us to make sure that the functions (A, B, H, C, C_1, C_2) are well behaved at $r = 0$. Notice that the components of the extrinsic curvature K_{ij} are also decomposed in a similar way (see below).

We also define the function $\psi = e^{\phi}$ and use it instead of ϕ in many expressions. For the lapse function we use the array $\alpha(r, z, t)$, while the shift in principle has three components: $\beta^r(r, z, t)$, $\beta^z(r, z, t)$, $\beta^\varphi(r, z, t)$.

Notice that when there is no angular momentum the metric can be simplified since in that case we have $\beta^\varphi = 0$, and $C_1 = C_2 = 0$. The code controls this using the logical parameter `angmom`: if this parameter is true then those arrays are turned on, while if it is false they are off. By default the parameter is false, so that those arrays are turned off.

It is also useful to define an inverse to the conformal factor as:

$$\chi := 1/\psi^n = \exp(-n\phi) \ , \tag{7.2}$$

with n a power controlled by the parameter `chipower`. For black hole spacetimes the conformal factor ϕ is singular at the black hole position, while χ remains regular, so it is best to evolve χ instead of ϕ. This can be controlled with the logical parameter `chimethod` (which by default is set to `.true.`).

7.2 Evolution equations

For the evolution equations, the code uses a Baumgarte-Shapiro-Shibata-Nakamura (BSSN) formulation adapted to axial symmetry. The specific form of the evolution equations used here can be found in [2].

There are two BSSN variants or "flavors" controlled by the parameter `bssnflavor`, which can take the values `lagrangian` or `eulerian` (the default is `bssnflavor=lagrangian`). They refer to the way in which the shift terms are added to make determinant of the conformal metric constant along time-lines (lagrangian), or constant along the normal-lines (eulerian).

There is also a parameter that allows one to switch off the evolution of the spacetime, which is useful in case one wants to evolve some matter field in a fixed background spacetime. The parameter is called `spacetime` and can have the values `dynamic` or `static` (default is `spacetime=dynamic`).

The main evolution variables (arrays) are:

`alpha`	The lapse function α.
`beta_r`	Contravariant shift component β^r.
`beta_z`	Contravariant shift component β^z.
`beta_p`	Contravariant angular shift component β^φ.
`phi`	The conformal factor ϕ (see equation 7.1).
`psi`	The conformal factor $\psi = e^\phi$ (see equation 7.1).
`A`	Component (r, r) of conformal metric $\tilde{g}_{rr} =$ `A`.
`B`	Component (z, z) of conformal metric $\tilde{g}_{zz} =$ `B`.
`H`	Component (φ, φ) of conformal metric $\tilde{g}_{\varphi\varphi} = r^2$ `H`.
`C`	Component (r, z) of conformal metric $\tilde{g}_{rz} = r$ `C`.
`C1`	Component (r, φ) of conformal metric $\tilde{g}_{r\varphi} = r^2$ `C1`.
`C2`	Component (z, φ) of conformal metric $\tilde{g}_{z\varphi} = r$ `C1`.

`trK` Trace of the extrinsic curvature $\texttt{trK} := K^m{}_m$.

`KTA` Component (r,r) of conformal trace-free extrinsic curvature $\tilde{K}^{TF}_{rr} = \texttt{KTA}$.

`KTB` Component (z,z) of conformal trace-free extrinsic curvature $\tilde{K}^{TF}_{rr} = \texttt{KTA}$.

`KTH` Component (φ,φ) of conformal trace-free extrinsic curvature $\tilde{K}^{TF}_{\varphi\varphi} = r^2\texttt{KTH}$.

`KTC` Component (r,z) of conformal trace-free extrinsic curvature $\tilde{K}^{TF}_{rz} = r\,\texttt{KTC}$.

`KTC1` Component (r,φ) of conformal trace-free extrinsic curvature $\tilde{K}^{TF}_{r\varphi} = r^2\texttt{KTC1}$.

`KTC2` Component (r,φ) of conformal trace-free extrinsic curvature $\tilde{K}^{TF}_{z\varphi} = r\,\texttt{KTC2}$.

`Delta_r` Contravariant component of auxilary BSSN variable Δ^r.

`Delta_z` Contravariant component of auxilary BSSN variable Δ^z.

`Delta_p` Contravariant component of auxilary BSSN variable Δ^φ.

Notice that, even though the auxiliary BSSN variables Δ^i are initially defined in terms of metric derivatives, they are later evolved independently with their own evolution equations (modified using the momentum constraints), as required in the BSSN formulation.

The code also calculates a series of auxiliary geometric quantities defined as:

`psi` The conformal factor $\texttt{psi} := \psi = e^\phi$. The code also calculates $\texttt{psi2} := \psi^2$ and $\texttt{psi4} := \psi^4$.

`chi` Another form of the conformal factor $\texttt{chi} = \chi = 1/\psi^n$. This function is in fact evolved instead of ϕ if one sets the logical parameter `chimethod=.true.` (by default it is true). This is useful for black hole evolutions as it improves the treatment of the central punctures.

8 Matter

9 Slicing conditions

10 Shift conditions

11 Initial data

The type of initial data is controlled by the character-type parameter `idata`. If you add a new type of initial data it should be appended to the list of allowed values for this parameter in the file `src/base/param.f90`. You should also add a corresponding call to your initial data routine in the file `src/base/initial.f90`.

12 Numerical methods

12.1 Time integration

For the time integration the code uses a method of lines, where the time integration and spatial differentiation are considered independent of each other.

13 List of main code parameters

14 Editing the code

15 Ollingraph

The code includes a simple 1D visualization package called "ollingraph". It is written in Python and uses Matplotlib to plot simple line plots and animations. At the moment it should work fine under python 3.11. It is supposed to have similar functionality to the old xgraph and ygraph packages, and expects the data files in the same format (see below).

The plots are quite simple, and meant for quick/dirty interactive visualization. These plots are not supposed to be used for figures intended for publication (use gnuplot or something similar

for that). To use the package type:

```
ollingraph file1 file2 ...  filen
```

When plotting several data files at once, they are all assumed to be of the same type. One can also add a custom name for the plot using the option –title:

```
ollingraph --title="Plot name" file1 file2 ...  filen
```

If this option is not there the plot is just named after the name of the data file being plotted, assuming there is only one, or just says "Multiple data files" if there is more than one file.

Before using it make sure that `ollingraph` has execution permission:

```
chmod +x ollingraph
```

There is also a package for simple plots of 2D arrays called `ollingraph2D`. It is called in a similar way:
`ollingraph2D file`

But in this case the file to be plotted must have extension `.2D`.

The data files are expected to have the following format:

- 0D files (those ending in .tl):

    1. One comment line starting with `#` or `"` with the file name (this line could be missing and it should still work).
    2. A series of lines with the data points, with the x and y values separated by blank spaces.

- 1D evolution-type files (those ending in .rl):

    1. Each time step begins with a comment line starting with `#` or `"` that contains the time in the format:

       ```
       #Time = (real number)
       ```

2. A series of lines with the data points, with the x and y values separated by blank spaces.
3. One or more blank lines to separate the next time step.

References

[1] "Regularization of spherical and axisymmetric evolution codes in numerical relativity", M. Ruiz, M. Alcubierre, D. Nuñez, Gen.Rel.Grav. **40**, 159-182 (2008); arXiv:0706.0923 [gr-qc].

[2] "Formulations of the 3+1 evolution equations in curvilinear coordinates", M. Alcubierre and M. D. Mendez, Gen.Rel.Grav. **43**, 2769 (2011); arXiv:1010.4013 [gr-qc].