Monday 24th April, 2023

SPHINCS_ID

Smoothed **P**article **H**ydrodynamics **IN** **C**urved **S**pacetime – **I**nitial **D**ata builder

A modular, object-oriented, OMP parallelized Fortran 2018 code to produce binary neutron stars initial data for SPHINCS_BSSN

User Manual for v1.8

Francesco Torsello

Contents

1 Introduction

1.1 Description of SPHINCS_ID

SPHINCS_ID is a modular, object-oriented, OMP parallelized Fortran 2018 code to produce initial data to be evolved in time with the General Relativistic, Lagrangian Hydrodynamics, Fortran 2018 code SPHINCS_BSSN [1], and the Newtonian, Lagrangian Hydrodynamics, Fortran code MAGMA2 [2].

Presently, SPHINCS_ID does not solve any equations for the initial data, but acts as an interface between an initial data solver and SPHINCS_BSSN or MAGMA2. It reads the data computed by the solver and produces the SPH and BSSN ID to be read and evolved in time with SPHINCS_BSSN or MAGMA2. Currently, it produces initial data for:

i. binary neutron star mergers and differentially rotating stars, using the data computed by the solvers within the C++ library LORENE [3, 4]

ii. binary systems of neutron stars, using the data computed by the FUKA solvers within the C++ library Kadath [5, 6]

iii. data on a Cartesian, uniform grid, representing a generic physical system

iv. Newtonian binary systems of neutron stars and white dwarfs, using the data computed by the TOV solver within `SPHINCS_BSSN`; in other words, two TOV stars are placed on an orbit given by the Newtonian 2-body problem

The modular and hierarchical structure of the code makes it easy to extend it to be able to set up initial data for other types of physical systems and other formulations of the Einstein equations (EE).

In `SPHINCS_ID`, each class is declared in its own module, and the implementations of its type-bound procedures are written in submodules. The present version consists of 3 main base classes: `idbase`, `particles` and `tpo`. `idbase` is an abstract class that represents a generic ID for any physical system (binary neutron star, differentially rotating star, ejecta, triple system of stars, etc...). The `particles` class represents the SPH particle distribution and its properties. The `tpo` class represents the $3+1$ decomposition of spacetime. The constructors of `particles` and `tpo` need an `idbase` object as one of their arguments, meaning that there cannot be objects of type `particles` or `tpo` without at least one `idbase` object. This makes sense since there cannot be particles if there is no fluid to model, and there cannot be a $3+1$ decomposition of the spacetime without a spacetime. The `tpo` class is abstract due to the following reason. The Einstein equations (EE), in numerical relativity (NR), can be posed as a Cauchy problem using different formulations. `SPHINCS_ID` should be able to easily and safely allow the programmer to implement such formulations, and the user to choose between them. `tpo` is meant to include all the properties shared by all $3+1$ formulations of the EE, and the properties of the standard $3+1$ formulation. A specific formulation is to be represented by a class that extends `tpo`. Version v1.8 of `SPHINCS_ID` has only the implementation of the Baumgarte–Shapiro–Shibata–Nakamura–Oohara–Kojima (BSSNOK, or just BSSN) formulation, represented by the class `bssn` that extends `tpo`. From now on, we will refer to the `bssn` class as representing also the `tpo` class, unless explicitly stated otherwise.

`idbase` is also an abstract class meant to contain all data and procedures shared by any type of ID. Specific IDs are to be represented by types that extend `idbase`. `idbase` defines the interface between all types of ID and the `particles` and `tpo` objects; this means that any variable or procedure that depends explicitly on some ID properties and is needed by the `particles` or `tpo` classes, must be a member of `idbase`. In the case of a procedure, if its implementation is the same for all possible physical systems, it should be implemented in a submodule of module `idbase`, and be a non-deferred (possibly non-overridable) member of `idbase`. If the implementation depends on the specific physical system, the procedure should be deferred to the extended type, and the implementation should be contained in a submodule of the module containing the declaration of the relevant extended type.

The classes `particles` and `bssn` are very much decoupled—in the sense that they never refer to each other internally (though some methods of `bssn` need an argument of type `particles`: for example, the one that computes the constraint violations on the mesh using the hydro data mapped from the particle distribution)—and orthogonal to each other—in the sense that they accomplish independent tasks—so that `SPHINCS_ID` can produce only SPH ID using only the `particles` class, or only BSSN ID using only the `bssn` class, or both. This is decided by the user by setting up the parameter files appropriately, as we will see in subsection 3.4.

As of v1.8, `SPHINCS_ID` has 4 programs: `sphincs_id_v1.8.x`, `convergence_test_v1.8.x`, `construct_newtonian_binary.x` and `write_par_eos_v1.8.x`. The first one produces SPH and/or BSSN ID for `SPHINCS_BSSN` or `MAGMA2`; the second one computes a Cauchy convergence test using the Hamiltonian and momentum contraints; the third one produces SPH and BSSN ID for `SPHINCS_BSSN` or `MAGMA2`, using data produced

by the TOV solver within SPHINCS_BSSN to place two TOV stars (neutron stars or white dwarfs) on an orbit determined by the Newtonian 2-body problem (conical section: ellipse, parabola or hyperbola; not straight line); the fourth one generates the parameter file par_eos.dat for single and piecewise polytropes to be used when running the LORENE executable init_bin to produce two TOV stars, as described in subsection 3.1.

The repository of SPHINCS_ID organizes the files in directories, each one containing a README.md file describing its content. Here follows a list of the directories, with a description of what they contain.

i. config contains the parameter files (or configuration files) needed by the programs provided by SPHINCS_ID

ii. mod is an empty directory needed during compilation. The *.mod files produced during compilation will be placed in it

iii. res contains resources related to SPHINCS_ID. Currently it stores files needed to produce the documentation with FORD (see below), and the User Manual.

iv. src contains the source files

v. tools contains scripts used to compile SPHINCS_ID, and the Project File needed by FORD to produce the documentation (see below)

1.2 Documentation of SPHINCS_ID

SPHINCS_ID is documented using FORD, which can be found at
`https://github.com/Fortran-FOSS-Programmers/ford`,
together with instructions on how to install and use it. Once FORD is installed, go to the root directory of SPHINCS_ID and type
`ford tools/documentation_sphincs_id.md`
in the SPHINCS_ID directory. This will generate the documentation in the subdirectory doc/. Open the file doc/index.html with any browser to consult the documentation.

The documentation is (temporarily?) hosted at
`https://sphincsid.bitbucket.io`.

2 Compilation of the codes

SPHINCS_ID v1.8 was compiled with and tested on:

i. GNU Fortran (gfortran) 10.2.1 20210110 and IFORT 19.1.0.166 20191121 compilers, on the r3x machines of the Department of Astronomy, Stockholm University

ii. GNU Fortran (gfortran) 8.3.0 and IFORT 2021.2.0 20210228, on the Sunrise HPC facility supported by the Technical Division at the Department of Physics, Stockholm University [7]

2.1 Compiling LORENE

A fork of LORENE, extended by Francesco Torsello to comply with the needs of the SPHINCS project, is needed to run SPHINCS_ID and can be found at
`https://bitbucket.org/sphincsid/lorene/src/master/`
The root directory contains a NOTICE file where a list of the modified and added source files is given. This fork also includes the patch check_fopen_error.patch and a slightly

different version of the patch `x_axe_limits.patch` from `LORENE2` (the difference is that, in the version needed by `SPHINCS_ID`, a parameter is hardcoded rather than being specifiable in a parameter file). `LORENE2` was developed by the developers of the Einstein Toolkit [8, 9], see

`https://bitbucket.org/einsteintoolkit/lorene/src/master/dist/`

Clone the repository wherever you prefer and set the environment variable `$HOME_LORENE` to the chosen path. The installation of the fork of `LORENE` needed by `SPHINCS_ID` proceeds in the same way as for the original `LORENE`, see the instructions at

`https://lorene.obspm.fr/install.html`

except that the user does not need to clone the original fork of `LORENE`, so the steps reported at

`https://lorene.obspm.fr/download.html`

can be skipped.

Two files `$HOME_LORENE/local_settings_r3x` and `$HOME_LORENE/local_settings_sunrise` are provided. Depending on the used host, the user should overwrite the file `$HOME_LORENE/local_settings` with one of them, before starting the compilation (as explained in the instructions in the original `LORENE` website). The file `$HOME_LORENE/local_settings_sunrise` was written by Mikica Kocic (member of the Technical Division at the Department of Physics).

2.2 Compiling `Kadath`

A fork of the `FUKA` branch of `Kadath`, extended by Francesco Torsello to comply with the needs of the SPHINCS project, is needed to run `SPHINCS_ID` and can be found at

`https://bitbucket.org/sphincsid/kadath/src/master/`

The root directory contains a `NOTICE` file where a list of the modified and added source files is given. Clone the repository wherever you prefer and set the environment variable `$HOME_KADATH` to the chosen path. The installation of this fork of `Kadath`, and of `FUKA`, proceeds in the same way as for the originals, read `$HOME_KADATH/README.md` for instructions (the user does not need to clone the original fork of `FUKA`, so skip that step in `$HOME_KADATH/README.md`). The code in `$HOME_KADATH/codes/bns_export` was added to `Kadath` by Francesco Torsello, and must be compiled in the same way as the other codes in `$HOME_KADATH/codes`.

To compile `Kadath` and `FUKA` on Sunrise, use the scripts in

`/cfs/home/pg/CHAP/compile-kadath-scripts/mpich`

written by Mikica Kocic (member of the Technical Division at the Department of Physics), and slightly modified by Francesco Torsello to compile also the code in `$HOME_KADATH/codes/bns_export`. Follow the instructions in the `README.md` in the same directory.

2.3 Compiling `SPHINCS_ID`

Modules and flavours. `SPHINCS_ID` v1.8 has several modules, possibly with several submodules. A "flavour" of `SPHINCS_ID` is defined as a successfully compiling set of its modules. `SPHINCS_ID` v1.8 has 4 flavours: *full*, *lorene*, *fuka*, and *interpolate*. The *full* flavour includes all the modules and links `SPHINCS_ID` to the `LORENE` and `Kadath` libraries; the *lorene* flavour includes only the modules needed to use the `LORENE` ID and generic ID on a Cartesian, uniform grid, and links `SPHINCS_ID` to the `LORENE` library; the *fuka* flavour includes only the modules needed to use the `LORENE` ID and generic ID on a Cartesian, uniform grid, and links `SPHINCS_ID` to the `Kadath` library; the *interpolate* flavour includes

only the modules needed to use a generic ID on a Cartesian, uniform grid.

The reason to allow for different flavours is that SPHINCS_ID should not necessarily be dependent on an ID solver to be compiled. For example if, in the future, LORENE won't be used anymore, there won't be any need to compile the modules related to it, nor to link SPHINCS_ID to the LORENE library.

The user decides which flavour to use by setting the appropriate parameters before the compilation, as described in the next paragraph.

Compilation. SPHINCS_ID uses procedures and variables defined in SPHINCS_BSSN, hence it must be linked to it. As of v1.8, SCons is used to for the building process; a SConstruct file is provided, which allows compilation with both the Intel and GNU Fortran compilers (gfortran, ifort). If SCons needs to be installed, follow the instructions at

https://scons.org/doc/production/HTML/scons-user/index.html.

As the first step to compile SPHINCS_ID, create a new directory and set the environment variable $HOME_SPHINCS to its path. Next, clone SPHINCS_ID and SPHINCS_BSSN into such directory with the commands:

```
git clone git@bitbucket.org:sphincsid/sphincs_id.git SPHINCS_ID
git clone git@bitbucket.org:SKR17/sphincs_repository.git SPHINCS_DynMetric
git clone git@bitbucket.org:SKR17/bssn_with_peter_diener.git BSSN
```

The names and the placement of these repositories can be changed by appropriately changing the paths in the SConstruct and src/SConscript files. We suggested to place the three repositories in the same directory since they are logically (and literally) linked.

SPHINCS_BSSN needs to be compiled first. This will produce the library which SPHINCS_ID links to. Using a Linux OS, the user compiles SPHINCS_BSSN by going to

$HOME_SPHINCS/SPHINCS_DynMetric/SPHINCS_BSSN,

modifying SConstruct to specify the local environment, and typing the command scons. Once the compilation is completed, the user should find a directory lib with the library inside it. Next, SPHINCS_ID is compiled by going to

$HOME_SPHINCS/SPHINCS_ID,

modifying SConstruct to specify the local environment, and typing the command scons. Note that the SConstruct file for SPHINCS_ID refers to the src/SConscript file, and the latter uses the *.py files in tools/. Hence, the user may need to modify some of them as well, when specifying the local environment. The object files will be placed in the newly-created build directory, and the executable files in the newly-created bin directory.

A number of options may be specified when compiling SPHINCS_ID:

i. flavour = {full_flavour = 1, lorene_flavour = 2, fuka_flavour = 3, interpolate_flavour = 4}
   The default flavour is full_flavour. Note that *full* links SPHINCS_ID to SPHINCS_BSSN, LORENE and Kadath libraries, and other libraries needed by LORENE (fftw3, blas, etc.); *lorene* links to the SPHINCS_BSSN and LORENE libraries and relative dependencies; *fuka* links to the SPHINCS_BSSN and FUKA libraries; *interpolate* only links to SPHINCS_BSSN.

ii. debug = {TRUE, FALSE}
   If TRUE, compile SPHINCS_ID with debug flags and link to the LORENE and FUKA debug libraries. The default is FALSE.

iii. fortran_compiler = {gfortran, ifort}. The default is ifort.

iv. compilers = {gnu, intel}. The option compilers sets the Fortran compilers, and, if used, it overrides the options fortran_compiler.There is no default value.

v. `verbose` = {`TRUE`, `FALSE`}
If `TRUE`, prints additional information during compilation. The default is `FALSE`.

vi. `host` = {`r3x`, `Sunrise`}
The machine on which `SPHINCS_ID` is compiled. Right now, only two hosts are supported. The host is automatically detected during compilation, and if it is not one of the two supported ones, the configuration for `r3x` is used. The user can add other options referring to the local host and environment.

All the defaults can be changed in `SConstruct`, so it is not needed to specify the options at each compilation. An example of compilation command with some options specified is:
`scons flavour=lorene_flavour fortran_compiler=ifort verbose=TRUE debug=TRUE`

3 Using the codes

3.1 Producing binary neutron star spectral initial data with `LORENE`

Read the reference [3] for a rather complete description of what `LORENE` does and how, to produce BNS ID.

`LORENE` provides two codes that can be used to produce BNS ID: `Bin_star` and `Binary_star`. Our experience is that `Bin_star` converges more easily to the solution. The codes are located at

`$HOME_LORENE/Codes/Bin_star`
`$HOME_LORENE/Codes/Binary_star`

From now on, we will consider `Bin_star` only. This code has been modified to comply with the needs of the `SPHINCS` project. Two executables are needed to produce BNS ID, `init_bin` and `coal_seq`, as described below. In order to produce them, go to the directory `$HOME_LORENE/Codes/Bin_star` and type `make init_bin` and `make coal_seq`. After that, they can be found in the same directory.

All the parameter files that will be mentioned in this section, and two examples of tabulated EOS that can be used with `LORENE` (one in `LORENE` format and one in CompOSE format), can be found in the directory

`$HOME_LORENE/Codes/Bin_star/Parameters/examples`

of the repository

`https://bitbucket.org/sphincsid/lorene/src/master/`

Step 1: Producing two TOV stars. In order to produce the two TOV stars to be used as the ID for the iteration that solves the constraints equations of General Relativity (GR), the user needs the following files:

i. the executable `init_bin`

ii. the parameter files:

(a) `par_grid1.d`, `par_grid2.d`. These specify the multi-domain spectral grids (one per star), namely:

i. the number of domains within a star (no more than 3)

ii. the number of domains outside the star

iii. the inner radii of each domain in units of the radius of the star, the first domain being a sphere and the others spherical shells; the last domain is compactified and extends to infinity. The last domain inside the star and the first domain outside the star should touch at the surface of the star, meaning that the inner radius of the first domain outside the star should be 1. It is desirable that the companion star is contained in a single domain—better if not the compactified one.

iv. the number of Chebyshev coefficients in each domain in the r, θ, ϕ directions, called `nr`, `nt` and `np`, respectively. These numbers determine the resolution of the spectral expansion, so more accurate results are obtained by increasing them. However, they have to be of the following form if using `FFT991`:

$$\texttt{nt} = 2^n 3^m 5^\ell + 1, \quad \text{with} \quad n \geq 1,\ m, \ell \geq 0,\ 2^n 3^m 5^\ell \geq 4, \tag{1a}$$
$$\texttt{np} = 2^n 3^m 5^\ell, \quad \text{with} \quad n \geq 1,\ m, \ell \geq 0,\ 2^n 3^m 5^\ell \geq 4, \tag{1b}$$
$$\texttt{nr} = 2^n 3^m 5^\ell + 1, \quad \text{with} \quad n \geq 1,\ m, \ell \geq 0,\ 2^n 3^m 5^\ell \geq 4, \tag{1c}$$

and of the following form if using `FFTW3`:

$$\texttt{nt} = 2n + 1, \quad \text{with} \quad n \geq 2, \tag{2a}$$
$$\texttt{np} = 2n, \quad \text{with} \quad n \geq 2, \tag{2b}$$
$$\texttt{nr} = 2n + 1, \quad \text{with} \quad n \geq 2, \tag{2c}$$

The user chooses `FFT991` or `FFTW3` in the file `$HOME_LORENE/local_settings`. `nt` and `np` are the same for each domain, and `nr` can be different for each domain. The last domain inside the star and the first domain outside the star should have the same `nr`, since the code smoothens the fields at the surface of the star, and in our experience the code complained during the smoothening if `nr` were different. See

`https://lorene.obspm.fr/Refguide/FFT991_2admissible__fft_8C_source.html`
`https://lorene.obspm.fr/Refguide/FFTW3_2admissible__fft_8C_source.html`

(b) `par_eos1.d`, `par_eos2.d`. These specify the EOS for each star, which can be a single or piecewise polytrope, or tabulated. See the `LORENE` documentation at

`https://lorene.obspm.fr/Refguide/classLorene_1_1Eos.html`

(and references to documentation therein) for details on how to specify the parameters in these files. The executable `write_par_eos.x` in `SPHINCS_ID` v1.8 produces this parameter file for single and piecewise polytropes. As mentioned before, examples of parameter files and tabulated EOS can be found in the directories

`$HOME_LORENE/Codes/Bin_star/Parameters/examples/tov_stars/par_eos_examples/`
`$HOME_LORENE/Codes/Bin_star/Parameters/examples/tabulated_eos_examples/`

of the repository

`https://bitbucket.org/sphincsid/lorene/src/master/`

(c) `par_init.par`. This parameter file specifies if the computation should be relativistic or Newtonian (choose relativistic if you want to use the ID with `SPHINCS_ID`), the separation in km between the centers of the stars (this can be left to 100km since the separation for the real BNS is specified later in another parameter file for the executable `coal_seq`, discussed in the next paragraph), the central enthalpies of the stars (which, together with the EOS, determine

mass and radius), the rotational state of the BNS (only the irrotational and corotational state can be specified, and they have to be the same for both stars), and if the conformal flatness assumptions has to be used (use it if you want to use the ID with `SPHINCS_ID`).

The output of `init_bin`, in addition to files for diagnostics, is the binary file `ini.d`. This is the file needed to run the next code, which produces the BNS ID.

Usually, one would like to produce TOV stars with a given (gravitational or baryonic) mass. The only way to do that in `init_bin`, at present, is to proceed by trial and error by changing the values of the central enthalpies. An efficient way is to find the desired mass using a low number of Chebyshev coefficients, so that the TOV equations are integrated very quickly; once the desired mass is found, the number of Chebyshev coefficients can be set to a higher value to get a more accurate solution.

Step 2: Using the TOV stars as ID to produce a BNS. Having the two TOV stars, the next step is to produce a sequence of BNS with different separations between the centers of the stars, with the possibility to specify desired baryonic masses, or desired gravitational masses (the gravitational masses in the binary systems, not for the isolated stars). The possibility to specify desired gravitational masses is not present in the original version of `LORENE`. Besides, the original version of `LORENE` allows specifying the initial separation, and then proceeds by reducing it by 5km until it finds a solution (usually, the closer the stars, the hardest it is for `LORENE` to converge). In our fork of `LORENE`, it is possible to specify the initial and final separations in units of 100km, and the separation step in km.

In order to solve the constraint equations of GR for a sequence of BNS configurations, the user needs the following files:

i. the executable `coal_seq`

ii. the binary file `ini.d` produced by the `LORENE` executable `init_bin`

iii. the parameter file `parcoal.d`. This specifies the name of the binary file containing the two TOV stars (`ini.d` is the default), the initial and final separations between the centers of the stars, the separation step, the desired baryonic masses and desired gravitational masses, the relaxation parameters and other parameters steering the iteration.

It is possible to ask `LORENE` to converge to a BNS having the required values for the baryonic masses of the stars, or the required values for the gravitational masses of the stars. If the baryonic masses are to be specified, set the parameters `mass_g_des1` and `mass_g_des2` to 0, and the parameters `mbar_voulue[0]` and `mbar_voulue[1]` to the desired values in solar masses. If the gravitational masses are to be specified, set `mass_g_des1` and `mass_g_des2` to the desired values, and `mbar_voulue[0]` and `mbar_voulue[1]` to the first guesses for the baryonic masses that would correspond to the desired value of the gravitational masses. Usually, such first guesses are $0.2M_\odot$ to $0.3M_\odot$ larger than the desired gravitational masses.

Regarding the relaxation parameters and the other parameters steering the iteration, unfortunately there is not much to say other than one has to find a set of parameters that leads to convergence, by trial and error. Parameters that worked for many cases are provided in the directory

`$HOME_LORENE/Codes/Bin_star/Parameters/examples/bns/`

of the repository

`https://bitbucket.org/sphincsid/lorene/src/master/`

The output of `coal_seq`, in addition to files that contain information and diagnostics, are binary files called `resu_*.d`, where the `*` stands for the separation between the centers of the stars in nits of 10km. These are the files needed by `SPHINCS_ID` as input.

Using the `CompOSE` database with `LORENE` and `SPHINCS_ID`. The `CompOSE` database [10] provides EOS that can be used with `LORENE`. See the section "Detailed description" at

`https://lorene.obspm.fr/Refguide/classLorene_1_1Eos__CompOSE.html`

to see how to use the `CompOSE` tables with `LORENE`.

Note that, in general, the downloaded tables do not provide data in β-equilibrium. This equilibrium is usually satisfied (or assumed to be satisfied) by a cold neutron star, or by two neutron stars in a binary system, but sufficiently far from each other—the latter being usually the system described with the BNS ID. In order to produce β-equilibrated data, the user can use the `CompOSE` software, which can be downloaded from

`https://compose.obspm.fr/software`

`https://compose.obspm.fr/manual`

We refer to Section 7.5 of the `CompOSE` Manual v3.00, for the instructions on how to use the software; also, the software itself gives guidelines on how to use it, during its own execution. The β-equilibrated EOS are stored in the files with extensions `*.nb.ns` and `*.thermo.ns`. The first contains the values of the baryon number density in fm^{-3} (which is the independent variable), the second contains several thermodynamical quantities as functions of the baryon number density. We refer to Section 4.2.2 of the `CompOSE` Manual v3.00 for more details on such thermodynamical quantities.

The original version of `LORENE`, as of 2023-04-21, reads in the files with extension `*.nb` and `*.thermo`, that is, those that are not β-equilibrated. The fork of `LORENE` modified to comply with the needs of the SPHINCS project, instead, reads in the files with extensions `*.nb.ns` and `*.thermo.ns`.

When producing ID for `SPHINCS_BSSN` with `SPHINCS_ID` using an ID file that needs a `CompOSE` tabulated EOS, the global path to the file `par_eos*.d` is needed to construct the EOS within `LORENE`. This path can specified in the parameter file `sphincs_id_parameters.dat` for `SPHINCS_ID` (note that, if this path must be used by `SPHINCS_ID`, the parameter `use_eos_from_id` must be set to `.FALSE.`).

3.2 Producing differentially rotating star spectral initial data with `LORENE`

Read [11] and references therein, for a description of what `LORENE` does and how, to produce DRS ID.

`LORENE` provides the code `rotdiff` to produce DRS ID, located at

`$HOME_LORENE/Codes/Rot_star`

The code has been extended to comply with the needs of the `SPHINCS` project. The executable `rotdiff` is needed to produce the ID, and can be obtained by going to the directory `$HOME_LORENE/Codes/Rot_star` and type `make rotdiff`. After compilation, the executable is found in the same directory.

All the parameter files that will be mentioned in this section, and two examples of tabulated EOS that can be used with `LORENE` (one in `LORENE` format and one in CompOSE format), can be found in the directory

`$HOME_LORENE/Codes/Rot_star/Parameters/Rotdiff`

of the repository

`https://bitbucket.org/sphincsid/lorene/src/master/`

In order to produce a differentially rotating star (DRS), the user needs the following files:

i. the executable `rotdiff`

ii. the parameter files:

   (a) `par_eos.d`. Same as for the BNS.

   (b) `parrotdiff.d`. This specifies: the multi-domain spectral grid for the star, as `par_grid1.d` does for the BNS; the relaxation and steering parameters, as `parcoal.d` for the BNS, but with options specific to the DRS.

3.3 Producing binary neutron star spectral initial data with `FUKA`

Read the reference [5] for a complete description of what `FUKA` does.

To use the `FUKA` solvers, follow the instructions in the README.md files at:
`https://bitbucket.org/fukaws/fuka/src/fukav2/codes/FUKAv2_Solvers/NS/`
`https://bitbucket.org/fukaws/fuka/src/fukav2/codes/FUKAv2_Solvers/BNS/` which can be found also in the local clone of (our fork of) `FUKA`, in the same directories.

3.4 Producing initial data with `SPHINCS_ID`

To produce SPH and BSSN ID with `SPHINCS_ID`, for `SPHINCS_BSSN` or `MAGMA2`, the user needs the following files:

i. the executable `sphincs_id_v1.8.x`

ii. the file containing the ID (whose location can be specified in the parameter file `sphincs_id_parameters.dat`, described below). The file must be renamed so that `sphincs_id_v1.8.x` knows what kind of data it stores; the first 5 characters of the name have to be one of these:

   (a) `BNSLO`: binary neutron star produced with `LORENE`

   (b) `DRSLO`: differentially rotating star produced with `LORENE`

   (c) `BNSFU`: binary neutron star produced with `FUKA`

   (d) `EJECT`: generic data on a Cartesian grid (the name `EJECT` will be deprecated; the first used data on a Cartesian grid was describing an ejecta, hence the name; it will be updated in a later version)

   Regarding the specific solvers:

   (a) When using `LORENE`, the binary file `resu_*.d` produced by the `LORENE` executable `coal_seq` is the needed one

   (b) When using `FUKA`, the pair of files `*.info` and `*.dat` produced by the `FUKA` executable `solve` from the BNS code, are the needed ones. Note that they must have the same name. In addition, the executable
   `$HOME_KADATH/codes/bns_export/bin/Release/export_bns`
   must be placed in the same directory as the ID files. This executable is called by `sphincs_id_v1.8.x` since `Kadath` is not thread-safe and cannot read the ID using OpenMP. Since `sphincs_id_v1.8.x` uses OpenMP, it cannot be linked to `Kadath` to read the ID in parallel. The current solution is that

`sphincs_id_v1.8.x` calls the executable `export_bns`, which reads the ID in parallel using MPI and prints it to ASCII files, one file per MPI rank. `SPHINCS_ID` then reads the data from these files in parallel using OpenMP and deletes the files afterwards.

(c) When using data on a Cartesian grid, the file with the data is the needed one. The currently supported format (developed to read the first used data describing an ejecta) consists of 8 columns containing: $(x, y, z, \rho, u, v_x, v_y, v_z)$. For clarity: the order matters; ρ is the baryon mass density; u is the specific internal energy; $\vec{v}$ is the spatial part of the fluid 4-velocity with respect to the Eulerian observer. A flat metric is assumed currently, but this can be easily changed by implementing the reading of more columns.

iii. the parameter files:

(a) `SPHINCS_fm_input.dat`, needed in `SPHINCS_ID` to specify what SPH kernel and EOS to use

(b) `gravity_grid_parameters.dat`, needed to specify the mesh features. For example, how many refinement levels should there be, how big should they be, what resolution should they have, and so on

(c) `bssn_parameters.dat`, needed to specify the finite-differencing (FD) order and the BSSN parameters

(d) `sphincs_id_parameters.dat`, needed to steer the execution of `SPHINCS_ID`. For example, in this parameter file the user specifies: How many BNS are to be set up? Which binary files containing the ID are to be used, and where are they stored? Which particle distribution to set up for each BNS? Should the particle positions be read from a formatted file? Should the SPH ID be set up, the BSSN ID, or both? How many output files to print? Where should they be stored? Should the constraint violations be computed? And other features

(e) `sphincs_id_particles.dat`, needed to specify parameters concerning the particle distributions. For example, in this parameter files the user specifies: How should the particles be placed within the stars? Should the Artificial Pressure Method (APM) be applied to them? Should the electron fraction Y_e be computed from the data provided by the `CompOSE` database? And other features

Note that only (d) and (e) are specific to `SPHINCS_ID`. (a), (b) and (c) are used by both `SPHINCS_BSSN` and `SPHINCS_ID`. The parameter files (d) and (e) contain descriptions and guidelines on how to set each parameter they contain.

3.4.1 Producing initial data for two TOV stars in a Newtonian binary system, with `construct_newtonian_binary.x`

Read [12, Ch. 3] and [13, Ch. III] for the formulation of the Newtonian 2-body problem, used in `construct_newtonian_binary.x` to produce the ID.

To produce SPH and BSSN ID with `construct_newtonian_binary.x`, for `SPHINCS_BSSN` or `MAGMA2`, the user needs the following files:

i. the executable `construct_newtonian_binary.x`

ii. the files containing the SPH and TOV ID for the TOV stars (whose location can be specified in the parameter file `newtonian_binary_parameters.dat`, described below). Note that the file `TOV.00000`, not the file `BSSN_vars.00000`, is needed

iii. the parameter files:

(a) `newtonian_binary_parameters.dat`, where: the location and names of input and output file are specified; the periastron parameter (which determines the periastron; see the parameter file itself), the initial distance in km between the stars, and the eccentricity of the orbit, are specified

(b) `gravity_grid_parameters.dat`, needed to specify the mesh features. For example, how many refinement levels should there be, how big should they be, what resolution should they have, and so on

3.5 Running a Cauchy convergence test

To run a Cauchy convergence test for the Hamiltonian and momentum constraints with `SPHINCS_ID`, the user needs the following files:

i. the executable `convergence_test_v1.8.x`

ii. the same parameter files as for `SPHINCS_ID`. The parameter file `sphincs_id_parameters.dat` has three parameters used only by `convergence_test_v1.8.x`. They are `numerator_ratio_dx`, `denominator_ratio_dx` and `ref_lev`. The latter specifies the refinement level used to do the Cauchy convergence test. The first two define the pairwise ratio (> 1) between the spacings of the three meshes used in the Cauchy convergence test (the numerator and denominator are individually used in the code; that is why they are separate parameters). The first spacing δ_1 is determined by the parameters specified in `gravity_grid_parameters.dat`. The others are $\delta_2 = \delta_1/r < \delta_1$ and $\delta_3 = \delta_2/r < \delta_2$, with $r :=$ `numerator_ratio_dx/denominator_ratio_dx` > 1. If the parameters in `sphincs_id_parameters.dat` and `sphincs_id_particles.dat` are set to use a SPH particle distribution, the Cauchy convergence test can be done also using the hydro data mapped from the particles to the mesh.

3.6 Producing the parameter file `par_eos.d` for `LORENE`

To produce the parameter file `par_eos.d` for `LORENE`, for single and piecewise polytropic EOS, run the executable `write_par_eos.x`. No parameter files are needed.

References

[1]S. Rosswog and P. Diener, "SPHINCS_BSSN: A general relativistic Smooth Particle Hydrodynamics code for dynamical spacetimes", Class. Quant. Grav. **38**, 115002 (2021).

[2]S. Rosswog, "The Lagrangian hydrodynamics code magma2", Monthly Notices of the Royal Astronomical Society **498**, 4230–4255 (2020), `https://doi.org/10.1093/mnras/staa2591`.

[3]E. Gourgoulhon, P. Grandclement, K. Taniguchi, J.-A. Marck, and S. Bonazzola, "Quasiequilibrium sequences of synchronized and irrotational binary neutron stars in general relativity: 1. Method and tests", Phys. Rev. D **63**, 064029 (2001).

[4]*LORENE: Langage Objet pour la RElativité NumériquE*, `https://lorene.obspm.fr/` (visited on 03/29/2023).

[5]L. J. Papenfort, S. D. Tootle, P. Grandclément, E. R. Most, and L. Rezzolla, "New public code for initial data of unequal-mass, spinning compact-object binaries", Physical Review D **104**, `10.1103/physrevd.104.024057` (2021), `https://doi.org/10.1103%2Fphysrevd.104.024057`.

[6] *Frankfurt University/Kadath (FUKA) Initial Data solver*, https://kadath.obspm.fr/fuka/# (visited on 03/29/2023).

[7] *HPC Sunrise cluster of the Department of Physics, Stockholm University*, https://it.fysik.su.se/hpc/index.html (visited on 03/29/2023).

[8] R. Haas, C.-H. Cheng, P. Diener, Z. Etienne, et al., *The Einstein Toolkit*, version The "Sophie Kowalevski" release, ET_2022_11, To find out more, visit http://einsteintoolkit.org, Oct. 2022, https://doi.org/10.5281/zenodo.7245853.

[9] F. Löffler, J. Faber, E. Bentivegna, T. Bode, P. Diener, R. Haas, I. Hinder, B. C. Mundim, C. D. Ott, E. Schnetter, G. Allen, M. Campanelli, and P. Laguna, "The Einstein Toolkit: A Community Computational Infrastructure for Relativistic Astrophysics", Class. Quantum Grav. **29**, 115001 (2012).

[10] *CompOSE: CompStar Online Supernovae Equations of State*, https://compose.obspm.fr/ (visited on 03/29/2023).

[11] E. Gourgoulhon, P. Haensel, R. Livine, E. Paluch, S. Bonazzola, and J. A. Marck, "Fast rotation of strange stars", 10.48550/ARXIV.ASTRO-PH/9907225 (1999), https://arxiv.org/abs/astro-ph/9907225.

[12] H. Goldstein, C. Poole, and J. Safko, *Classical Mechanics* (Addison Wesley, 2002).

[13] L. Landau, E. Lifshitz, J. Sykes, and J. Bell, *Mechanics: Volume 1*, Course of theoretical physics (Elsevier Science, 1976).