Wednesday 7th September, 2022

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 v1.6

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

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. The code is currently under heavy development.

In `SPHINCS_ID`, each class is declared in its own module, and the implementations of its procedures are written in submodules. The present version consists of 3 main base classes: `idbase`, `particles` and `tpo`. `idbase` is an abstract class that represent 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 $3+1$ formulation 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 physical system, and there cannot be a $3+1$ formulation of the spacetime without a physical system as well. The `tpo` class is abstract since the spacetime, in numerical relativity (NR), can be evolved using different formulations of the Einstein equations (EE), and `SPHINCS_ID` should be able to easily and safely allow the programmer to implement them, and the user to choose between them. Hence, the specific formulation used is described by a class that extends `tpo`. In other words, `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. Version v1.6 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. `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 id_base, 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 contained in a submodule of the module containing 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`)—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 v1.6, `SPHINCS_ID` has 3 programs: `sphincs_id_v1.6.x`, `convergence_test_v1.6.x` and `write_par_eos_v1.6.x`. The first one produces SPH and/or BSSN ID for `SPHINCS_BSSN`; the second one computes a Cauchy convergence test using the Hamiltonian and momentum contraints; the third generates the parameter file `par_eos.dat` for piecewise polytropes to be used when running the `LORENE` executable `init_bin` to produce two TOV stars, as described in subsection 3.1.

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, type

```
ford documentation_sphincs_lorene.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 currently hosted at
https://francescotorsello.github.io/SPHINCS_ID-doc/index.html.

2 Compilation of the codes

The libraries LORENE and Kadath, their codes, and SPHINCS_ID v1.6 were compiled and tested with the Intel and GNU compilers, on the r3x machines of the Department of Astronomy, Stockholm University, and on the Sunrise HPC facility supported by the Technical Division at the Department of Physics, Stockholm University [7].

2.1 Compiling LORENE

A modified version of LORENE is needed to run SPHINCS_ID, and can be found at
https://bitbucket.org/ftorsello/lorene/src/master/
The installation of this modified version of LORENE proceeds in the same way as for the original LORENE, see the instructions at
https://lorene.obspm.fr/install.html

Two files $HOME_LORENE/local_settings_r3x and $HOME_LORENE/local_settings_sunrise are provided. Depending on the used host, one of them should overwrite the file $HOME_LORENE/local_settings before starting the compilation (as explained in the instructions). 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

An extended version of Kadath is needed to run SPHINCS_ID, and can be found at
https://bitbucket.org/ftorsello/kadath/src/link/
The installation of the extended version of Kadath, and of FUKA, proceeds in the same way as for the originals, read $HOME_KADATH/README.md for instructions. The code in $HOME_KADATH/codes/bns_export was added to Kadath, 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
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 flavors. SPHINCS_ID v1.6 has 16 modules, each one with several submodules. A "flavor" of SPHINCS_ID is defined as a successfully compiling set of its modules. SPHINCS_ID v1.6 has 4 flavors: *full*, *lorene*, *fuka*, and *interpolate*. The *full* flavor includes all the modules and links SPHINCS_ID to the LORENE and Kadath libraries; the *lorene* flavor 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* flavor 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* flavor includes only the modules needed to use a generic ID on a Cartesian, uniform grid.

The reason to allow for different flavors is that SPHINCS_ID should not necessarily be dependent on an ID solver to be compiled. If in the future, LORENE won't be used anymore, there is no need to compile the modules related to it, nor to link SPHINCS_ID to the LORENE library.

The user decides which flavor to use by setting the appropriate parameters at compile time, as described in the next paragraph.

Compilation. SPHINCS_ID uses parts of SPHINCS_BSSN, hence it has to be compiled together with it. As of v1.6, SCons is used to compile the code; a SConstruct file is provided, which allows compilation with both the Intel and GNU C++ (gcc, g++, icpc) and 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, SPHINCS_ID and SPHINCS_BSSN needs to be appropriately placed inside $HOME_SPHINCS:

$HOME_SPHINCS/BSSN
$HOME_SPHINCS/sphincs_repository

The directories $HOME_SPHINCS/sphincs_repository and $HOME_SPHINCS/BSSN contain clones of two separate repositories which can be found, respectively, at

https://bitbucket.org/ftorsello/sphincs_repository_ft/
https://bitbucket.org/ftorsello/bssn_ft

A version of SPHINCS_BSSN and SPHINCS_fix_metric is provided in the first of this repositories, as of v1.6 of SPHINCS_ID. This is because there are some minor modifications to them, with respect to their latest versions. You need to use these versions to run SPHINCS_ID. Hopefully this will change soon. The same is true for the second repository.

On a machine running Linux, the user compiles SPHINCS_ID by going to
$HOME_SPHINCS/sphincs_repository/SPHINCS_ID,
where the SConstruct file is placed, and typing the command scons.

A number of options may be specified when compiling SPHINCS_ID:

i. flavor = {full_flavor = 1, lorene_flavor = 2, fuka_flavor = 3, interpolate_flavor = 4}
The default flavor is full_flavor. Note that *full* links SPHINCS_ID to both the LORENE and FUKA libraries, and other libraries needed by LORENE (fftw3, blas, etc.); *lorene* links only to the LORENE library and relative dependencies; *fuka* links only to the FUKA library; *interpolate* does not link to any library.

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. cpp_compiler = {gcc, g++, icpc}. The default is icpc.

v. compilers = {gnu, intel}. The option compilers sets both the C++ and Fortran compilers, and, if present, it overrides the options fortran_compiler and cpp_compiler. There is no default value.

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

vii. `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.

All the defaults can be changed in the `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 cpp_compiler=g++
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/ftorsello/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 to place the following files in the same directory:

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:

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

$$\texttt{np} = 2^n 3^m 5^\ell, \quad \text{with} \quad n \geq 2,\ m, \ell \geq 0, \tag{1b}$$

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

where `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. The formulas in (1) are necessary because `LORENE` uses a Fast Fourier Transform algorithm that needs them.

(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` v1.6 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/ftorsello/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 irrotational and corotational can be specified separately for each star; irrotational state is often preferable since, in a BNS, the stars are not usually tidally locked), 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 the desired value to get a more accurate solution.

Step 2: Using the TOV stars as ID to produce a BNS. Having the two TOV stars, we 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 version of `LORENE`, it is possible to specify in a parameter file the initial and final separations, and the separation step in km.

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

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.

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/ftorsello/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 km. These are the files needed by `SPHINCS_ID` as input.

Using the `CompOSE` database with `LORENE`. The `CompOSE` database [8] provides many 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`

Note that the `CompOSE` software has to be used also to produce the `*.beta` file containing the $Y_e(n_{\rm b})$ data, that is, the electron fraction as a function of the baryon number density

(* stands for the name of the EOS). As of v1.6, this *.beta file is used by SPHINCS_ID to linearly interpolate Y_e at the particle positions, based on the values of $n_{\rm b}$ at such positions.

3.2 Producing differentially rotating star spectral initial data with LORENE

Read [9] 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/ftorsello/lorene/src/master/

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

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 at:

https://bitbucket.org/fukaws/fuka/src/fukav2/codes/FUKAv2_Solvers/NS/

https://bitbucket.org/fukaws/fuka/src/fukav2/codes/FUKAv2_Solvers/BNS/

3.4 Producing initial data with SPHINCS_ID

To produce SPH and BSSN ID with SPHINCS_ID, for SPHINCS_BSSN, the user needs to place in the same directory the following files:

i. the executable sphincs_id_v1.6.x

ii. the file containing the ID (this does not need to be placed in the same directory, and its location can be specified in the parameter file sphincs_id_bns_parameters.par, described below). The file must be renamed so that sphincs_id_v1.6.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` is 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_test`
must be placed in the same directory as the ID files. This executable is called by `sphincs_id_v1.6.x` since `Kadath` is not thread-safe and cannot read the ID using OpenMP. Since `sphincs_id_v1.6.x` uses OpenMP, it cannot be linked to `Kadath` to read the ID in parallel. The current solution is that `sphincs_id_v1.6.x` calls the executable `export_bns_test`, which reads the ID in parallel using MPI. The name `export_bns_test` will be changed in a later version.

(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_bns_parameters.par`, 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_bns_particles.par`, 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 (e) and (f) 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.

3.5 Running a Cauchy convergence test

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

i. the executable `convergence_test_v1.6.x`

ii. the same parameter files as for `SPHINCS_ID`

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

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

3.7 Initial data library

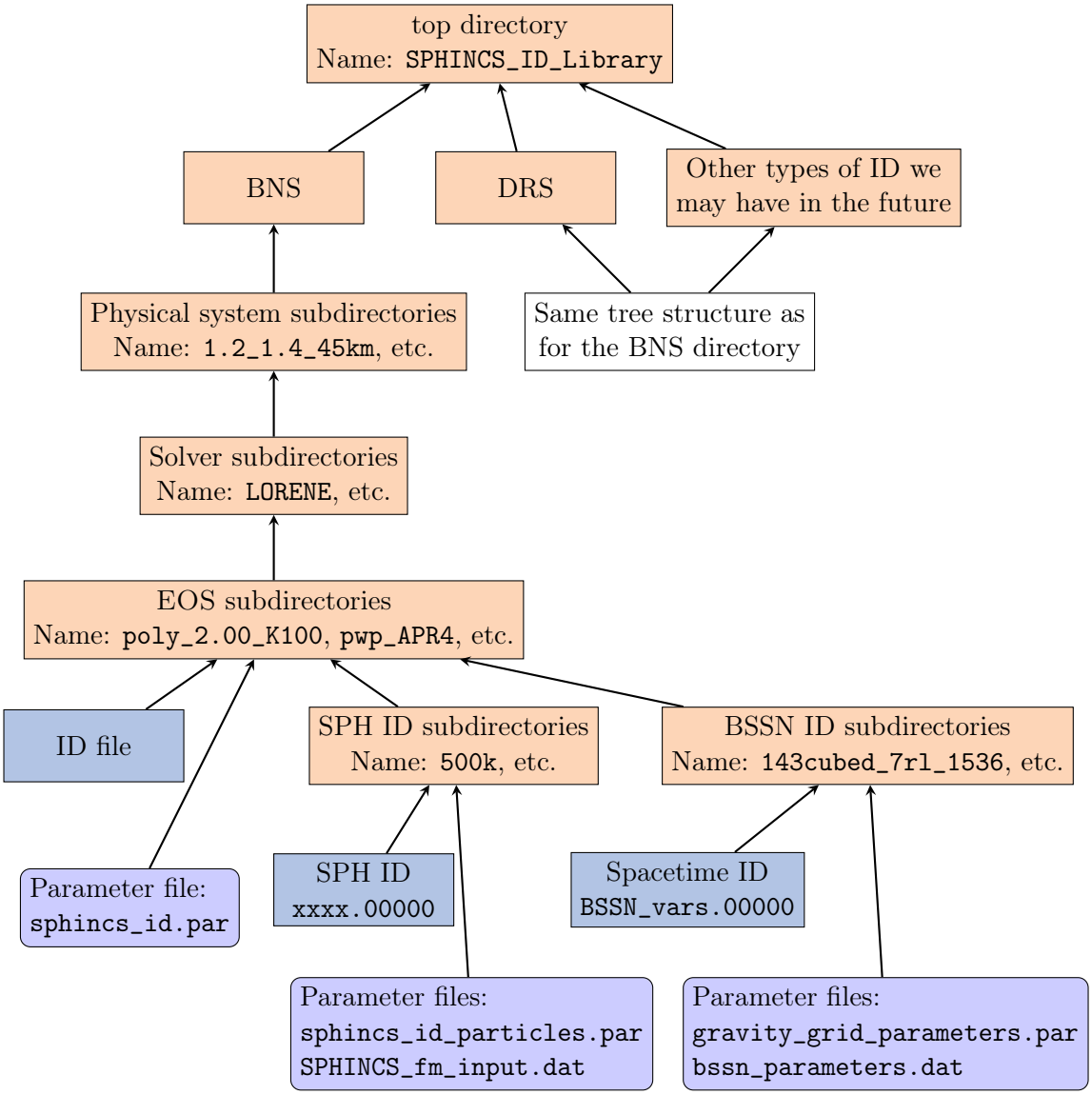

Figure 1: Directory structure for the ID library produced with `SPHINCS_ID`. Orange rectangles represents one or more directories; blue rectangles are ID files; purple rounded rectangles are executable and parameter files; white rectangles are plain text.

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/.

[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/#.

[7] *HPC Sunrise cluster of the Department of Physics, Stockholm University*, https://it.fysik.su.se/hpc/index.html.

[8] *CompOSE: CompStar Online Supernovae Equations of State*, https://compose.obspm.fr/.

[9] 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.