SPHYNX instructions
version 1.1: pages 1–4
https://astro.physik.unibas.ch/sphynx

Using the hydrodynamics code SPHYNX

Rubén M. Cabezón[1,2] and Domingo García-Senz[3,4]

Abstract
This document describes the use of the Smoothed Particle Hydrodynamics (SPH) code for astrophysical applications SPHYNX. The code can be downloaded here. In the following we introduce the main structure of SPHYNX, its parameters, and how to set it up for some test cases. We assume that the reader has knowledge on the SPH technique.

1 Introduction

SPHYNX* is of Newtonian type and grounded on the Euler-Lagrange formulation of the smoothed-particle hydrodynamics technique. Its distinctive features are: the use of an integral approach to estimating the gradients; the use of a flexible family of interpolators called *sinc* kernels, which suppress pairing instability; and the incorporation of a new type of volume elements which provides a better partition of the unity. Unlike other modern formulations, which consider volume elements linked to pressure, our volume element choice relies on density. SPHYNX is, therefore, a density-based SPH code. For a detailed review on the implementation of the SPH equations in SPHYNX, we refer the reader to Cabezon et al. (2017).

2 Parallelization

SPHYNX is an hybrid MPI+OpenMP code written in Fortran90. The main idea behind this implementation is that each MPI process is located in a different node, while the OpenMP threads use the available CPUs on each node. Different process/thread distributions are also possible, depending on the computer/cluster topology and resources. The amount of MPI processes can be controlled from the `parameters.f90` file via the parameter `nodes`, while the number of threads is controlled by the environment variable OMP_NUM_THREADS.

3 Code structure

The basic version of SPHYNX includes four directories:

- `compile/` where the `makefile` is located and the compilation of the code is done.
- `sphynx/` where the source code resides.
- `initialmodels/` where the input files for the particles distribution and properties should be located.
- `run/` where the output of the simulation is stored, and where the initial conditions are given. `run/` must have a subdirectory called `data/` where the particles distribution and properties are stored at certain intervals defined in `parameters.f90`. The folder `run/` can be renamed to something more descriptive, but then the `makefile` should be modified accordingly (see below).

4 Compilation

SPHYNX only needs a FORTRAN90 compiler and an MPI library. In its current version, SPHYNX can be compiled and start to run out of the box with Intel Fortran Compiler 2017 and OpenMPI 2.0.1, and also with Gfortran 5.4.0. Newer versions of the compiler and the MPI library might require some modifications that will we released in the future. We will additionally release support for other FORTRAN compilers.

In order to compile SPHYNX, go to `compile/` and do:

```
make clean
make -j
```

This will create an executable in the simulation directory. You can change the makefile to adapt this to your needs. In particular, the variable `path` points to the location of the SPHYNX source code, `pathsim` points to the location of the simulation directory, and `executable` is the name of the final executable produced by the `makefile`. The standard compilation flags are stored in the variable `SF` and `SF2`. If needed, debug flags can be used instead via the variables `DF` and `DF2`. The default compiler is `intel`, but this can be changed to use Gfortran at the beginning of the `makefile`, setting `compiler = gnu`.

5 The source code

See Table 2 for a detailed list of the files that are located in the `sphynx/` and `run/`directories, and for what are they used. SPHYNX was designed trying to minimize the amount of changes in the source code, so that the user needs

[1]Scientific Computing Core, sciCORE, Universität Basel, Switzerland
[2]Departement Physik, Universität Basel, Switzerland
[3]Departament de Física, Universitat Politècnica de Catalunya, Barcelona, Spain
[4]Institut d'Estudis Espacials de Catalunya, Barcelona, Spain

Corresponding author:
Rubén M. Cabezón - sciCORE, Universität Basel
Klingelbergstrasse, 61 - 4056 Basel, Switzerland
Email: ruben.cabezon@unibas.ch

*Before you ask, SPHYNX stands for nothing. It is just a cool word with SPH in it! If you really want to find a meaning to it, you can think that it comes from Smoothed Particle HYdrodyNamics eXtended, but this has as much meaning as Simple Phrase Handmade to Yield a Naive teXt.

to change only those files located in the simulation folder. Nevertheless, experienced users might want to modify the internal machinery of SPHYNX.

6 The parameter file

This is a very important file as it contains all the parameters and global quantities that control the simulations. Any variable declared here, will be accessible from any module or subroutine that does `import parameters`. Most of the files in SPHYNX import this module, so changes here affect the whole code. In Table 3 we detail the most relevant variables and parameters.

7 Implementation considerations

The following points are important remarks to keep in mind when using SPHYNX:

- 3D vectors like position, velocity, gradient of pressure, etc... are stored in 1D arrays of length 3×`nmax` in the form $(x_1, ..., x_n, y_1, ..., y_n, z_1, ..., z_n)$, being n the number of particles in the simulation.
- Each particle has a fixed ID (from 1 to n).
- The initial conditions of the simulated scenario can be defined in `init_scenario.f90`.
- For computational convenience, the whole system is shifted to the positive quadrant of the dimensional space, after initial conditions have been defined. This is automatically done by SPHYNX via finding the global size of the simulated scenario and displacing all particles ≈ 5 times that size. This is stored in the variable `despl`. This is automatically subtracted when outputting data.
- The simulation scenario should contain a `/data` directory.
- SPHYNX uses CGS units by default.

8 Getting started

In the following we describe two 3D tests that can be done, out of the box, with the data provided in the SPHYNX website.

Evrard Collapse

A relevant test in Astrophysics is the numerical study of the gravitational collapse of a gaseous configuration, also known as the Evrard test (Evrard 1988). This test includes gravity and has been used many times to check hydrodynamics codes (see for example Springel (2010) and references therein). See also the main SPHYNX paper (Cabezon et al. 2017) for more details on this test and results. This is the default configuration of SPHYNX after downloading it. Therefore, it just needs to be compiled (see Sec. 4) and launched from the `evrard/` directory:

```
mpirun -np <MPI_processes> evardtest
```

The usage of OpenMP is optional, but highly recommendable. In order to control the number of threads, you can simply export the corresponding environment variable before launching the simulation:

```
export OMP_NUM_THREADS=<number_of_threads>
```

Be aware, that the variable `nodes` is set to 8. This means that 8 MPI processes will be launched. You might need to change these values to adequate them to your computational resources[†] and recompile before launching the simulation.

In its default configuration, SPHYNX will generate a series of output files listed on Table 1:

Table 1. List of output files.

File name	Content
Simulation directory (`evrard/`)	
`conservelaws.d`	Conservation laws
`estabil.d`	Central density and radius evolution
`REPORT`	Summary of parameters
`outputtimes.d`	Physical time for each output file
`timectrl.d`	Timestep evolution
`timing.d`	Time profiling
`dumpfile<iteration>`	Restart files
Output data directory (`data/`)	
`s1.<iteration>`	Particle data for `<iteration>`

The contents and formats of each file are described in the following:

- `conservelaws.d`: time, kinetic energy, internal energy, gravitational energy, total energy, total linear momentum, total angular momentum.
`format(7(1x,es17.10))`
- `estabil.d`: time, max. radius, central density.
`format(3(1x,es23.16))`
- `outputtimes`: iteration, time dumpfile, time standard file
- `timectrl.d`: Courant timestep, density timestep, acceleration timestep, used timestep, time
`format(10(1x,es12.5))`
- `timing.d`: Tree building, neighbors finding, neighbors comm., density and h update, density comm., IAD_0 terms, IAD_0 terms comm., EOS, divergence and curl of velocity, divergence and curl of velocity comm., moment and energy eqs., moment and energy eqs. comm., gravity, gravity comm., update, timestep evaluation, conservation laws, output, total time, total time for comm.

All `.d` files are opened with `POSITION=append`, meaning that one row of data is appended each timestep. Be aware that if a new simulation is to be started in the same folder as a previous one, these `.d` files should be deleted.

The contents of the `s1.<iteration>` and `dumpfile<iteration>` files can differ among simulations, but in general they have the following structure:

- `s1.<iteration>`: particle ID, x, y, z, smoothing length, specific internal energy, density, v_x, v_y, v_z, radius, $|\nabla \times v|$,pressure, $|\nabla v|$, grad-h, f_x, f_y, f_z, ∇P_x, ∇P_y, ∇P_z,temperature, $\dot{u}^{AV}$, $\dot{u}$, mass inside $2h$,

[†]Also, keep in mind that the total number of CPUs that you will use is the number of MPI processes times the number of OpenMP threads that you have defined in your environment.

volume element, radial velocity, number of neighbors, min. distance among neighbors, mass coordinate, gravitational potential, speed of sound, sinc kernel index, volume normalization.
`format(1x,i7,34(1x,es12.5))`

- `dumpfile<iteration>`: follow the same structure as `s1.<iteration>`, but save the date with higher precision to allow accurate restart, if needed.
  `format(1x,i7,34(1x,es23.16))`

The user can change the output contents in `outputmod.f90`. The standard formats for the `s1.` and `dump` files are defined in the `parameters.f90` file as `formatout` and `formatdump`, respectively.

Wind-bubble interaction

Popularly known as the 'blob' test (Agertz et al. 2007), this problem has challenged SPH codes for a long time. The basic setting of this test gathers many pieces of physics, such as strong shocks and mixing due to the KH and RT instabilities in a multi-phase medium with a large density contrast. The initial configuration consists in a dense spherical cloud of cold gas embedded in a hotter ambient medium. The cloud is initially at rest while the ambient background (the wind) moves supersonically. The wind-cloud interaction generates a bow shock and, in short time, the cloud is fragmented and mixed with the background owing to the combined effect of ablation and KH and RT instabilities.

For this example, we will re-use and modify the source files from simulation folder (`evrard/`) of the Evrard collapse simulation, in order to crate a new simulation named `windblob/`. This test case can be used as an example to prepare any other simulation using the basic SPHYNX files as a starting point.

- Download from the SPHYNX website the initial model (`InitWindBlob.dat`) and put it in the `initialmodels/` folder.
- Create a folder named `windblob/` and copy inside all the `.f90` files from the `evrard/` folder.
- Create a data subdirectory (`windblob/data/`)
- Modify `parameters.f90` with the following changes:

```
nmax = 3157385
nnl = 20000
ncubes = 27
xbox=0.25d0
ybox=0.25d0
maxtstep=1.d-3
gravity = .false.
deltaini = 1.d-9
inivel0 = .false.
inputfile='InitWindBlob.dat'
```

- Still in `parameters.f90`, create a new double precision variable: `masscloudinic`. Save the changes and close the file.
- Open `readdata.f90` and change the first `do`-loop to:

```
masscloudinic=0.d0
do i=1,n
   read(1,'(13(1x,es12.5))') a(i),a(i+n),a(i+n2),&
   &v(i),v(i+n),v(i+n2),promro(i),p(i),h(i)
   outofNRup(i)=.false.
   outofNRdw(i)=.false.
   temp(i)=1.d0
   mue(i)=2.d0
   mui(i)=10.d0
   masa(i)=1.99915d-08
   if(promro(i).gt.2.d0) then
      masscloudinic=masscloudinic+masa(i)
   endif
   u(i)=p(i)/gamma/promro(i)
   c(i)=sqrt(2.d0/3.d0*u(i))
   ballmass(i)=promro(i)*h(i)**3
enddo
```

- Save the changes and close the file.
- Open `outputmod.f90` and add the following to the header:

```
DOUBLE PRECISION masscloud
DOUBLE PRECISION,PARAMETER:: rhocloud=10.d0
DOUBLE PRECISION,PARAMETER:: uambient=1.5d0
DOUBLE PRECISION,PARAMETER:: tkh=0.0937d0
```

- After the header, add the following lines:

```
masscloud=0.d0
do i=1,nmax
   if(promro(i).ge.0.64d0*rhocloud.and.&
      &u(i).le.0.9d0*uambient) then
      masscloud=masscloud+masa(i)
   endif
enddo
open(2,file='masscloud.d',position='append')
write(2,'(3(1x,es12.5))') tt/tkh,masscloud,&
   &masscloud/masscloudinic
close(2)
```

- Save the changes and close the file.
- Finally, go to the compilation folder `compile/` and edit `makefile`.
- Change the variables `pathsim` and `executable` to:

```
pathsim = ../windblob/
executable = windblob
```

- Save the changes and close the file.

After these changes, we have prepared the parameters for a simulation without gravity and with periodic boundary conditions. We have also set the parameters and read modules to input the data from the initial model. A final change was done in the output module to create an additional data file, where the surviving fraction of cloud is tracked. The model should run after successfully compiling.

These are reasonably standard steps that can be used to setup your own simulation. In more complex scenarios, where the initial model does not provide all initial conditions, these should be specified in `init_scenario.f90`.

9 Disclaimer

The authors release the SPHYNX code "AS IS". We don't claim that SPHYNX will solve your scientific questions - if it does, great! If SPHYNX doesn't work or doesn't do it as you want: tough. If you lose a million because SPHYNX messes up, it's you that's out of the million, not us. If you don't like this disclaimer: tough. We reserve the right to do the absolute minimum provided by law, up to and including nothing. This is basically the same disclaimer that comes with all software packages, but ours is in plain English and theirs is in legalese. We didn't really want to include a disclaimer at all, but our lawyers insisted. We tried to ignore them, but they threatened us with the shark attack at which point we relented.

References

Agertz, O., Moore, B., Stadel, J., et al. 2007, MNRAS, 380, 963

Cabezon, R. M., Garcia-Senz, D., & Figueira, J. 2017, A&A, 606, A78

Evrard, A. E. 1988, MNRAS, 235, 911

Springel, V. 2010, MNRAS, 401, 791

Table 2. List of files of the source code SPHYNX.

File name	Purpose
	`sphynx/` directory
`apply_PBC.f90`	Calculates the displacement necessary to apply periodic boundary conditions.
`buildtreemod_grav.f90`	Creates the octal tree and calculates the center of mass in each cell.
`calculate_avglogrho.f90`	Calculates the averaged log_{10}(density) within 2h.
`calculate_density.f90`	Calculates the density and the terms needed for the VEs, grad-h and grad-n.
`calculate_divv.f90`	Calculates the divergence and curl of the velocity field.
`calculate_hNR.f90`	Newton-Raphson to update the smoothing length.
`calculate_hNR_eq.f90`	Newton-Raphson to update the smoothing length and the kernel index of the $sinc$ family.
`calculate_hpowers.f90`	Calculates several powers of the smoothing length.
`calculate_IAD.f90`	Calculates the matrix terms needed to use the IAD_0 formalism.
`calculate_norm.f90`	Calculates the normalization factor of the $sinc$ kernels.
`calculate_omega.f90`	Calculates the grad-h terms and the Volume Elements.
`conservemod.f90`	Checks and stores conservation laws.
`cubicspline.f90`	Provides the factors and normalization needed to evaluate the cubic spline kernel.
`eosid_t.f90`	Ideal equation of state using temperature as input.
`eosid_u.f90`	Ideal equation of state using internal energy as input.
`eosmod.f90`	Equations of State module.
`estabilmod.f90`	Tracks central density and maximum radius. Creates an array of indices sorting particles by radii.
`findneighbors.f90`	Creates a list of indices of the neighbors for each particle.
`indexx.f`	Returns an integer array that sorts the input array from small to big. Based in Numerical Recipes.
`init.f90`	Initial values for several code variables.
`masscenter.f90`	Calculates the center of mass of the particles distribution.
`momeqn.f90`	Calculates SPH momentum and energy equations including IAD_0 terms and artificial viscosity.
`nomfils.f`	Creates names of files from iteration numbers.
`printreport.f90`	Outputs main parameters used in the simulation.
`profile.f90`	Times sections of the code.
`reinit.f90`	Re-initialization of variables after each iteration.
`sinx_x.dat`	Table of $\operatorname{sinc} x$ values to interpolate the $sinc$ kernel.
`sphynx_hybrid.f90`	Main code of SPHYNX.
`testparameters.f90`	Checks consistency of input parameters.
`timectrlmod.f90`	Calculates next timestep.
`treewalkmod_grav_mefec.f90`	Calculates 3D gravitational force and potential up to the quadripolar term.
`update.f90`	Updates particles position, velocities and internal energy.
`wkernel.f90`	SPH interpolation kernel.
	`run/` (i.e. simulation) directory
`init_scenario.f90`	Initialization of several code and numerical model variables.
`outputmod.f90`	Output of data. Assumes that `./data/` directory exists.
`parameters.f90`	Main parameter module that includes all parameters and global variables.
`readdata.f90`	Input of data. Assumes that the input file is located in `../initialmodels/`

Table 3. List of main variables and parameters in `parameters.f90`.

Variable name	Type	Purpose	Parameter name	Type	Purpose
`a`	REAL(dim$\times$nmax)	Particles positions	`alfa`	REAL	Artificial viscosity (AV) constant
`ac`	REAL(dim$\times$nmax)	Particles accelerations	`balsara`	LOGICAL	Activate Balsara correction
`avisc`	REAL(nmax)	du/dt due to the AV	`conduction`	LOGICAL	Activate thermal conduction
`c`	REAL(nmax)	Speed of sound	`comments`	CHARACTER	User-defined comments
`cm`	REAL(dim)	Center of mass	`conserve`	LOGICAL	Activate output of conservation laws
`cube`	INTEGER(ALLOC)		`deltahindex`	INTEGER	Max. range of kernel index
`dt`	REAL	Timestep	`deltaini`	REAL	Initial timestep
`energy`	REAL(nmax)	du/dt	`dim`	INTEGER	Dimensionality
`f`	REAL(dim$\times$nmax)	Gravitational force	`dkacc`	REAL	Max. allowed change in acceleration
`gradh`	REAL(nmax)	Grad-h term	`dkro`	REAL	Max. allowed change in density
`gradn`	REAL(nmax)	Grad-n term	`dumpdelay`	INTEGER	Interval of iterations to write a dumpfile
`gradp`	REAL(dim$\times$nmax)	Gradient of pressure	`eos`	INTEGER	Selects the EOS
`h`	REAL(nmax)	Smoothing length	`equalization`	LOGICAL	Activate equalization
`h2,h3,...`	REAL(nmax)	Powers of h	`estabil`	LOGICAL	Activate output of central density
`hd`	REAL(nmax)	2h	`flags`	LOGICAL	Activate verbosity
`hm1,hm2,...`	REAL(nmax)	Inverse powers of h	`formatdump`	CHARACTER	Format of dumpfiles
`id`	INTEGER	MPI process ID	`formatin`	CHARACTER	Format of input file
`indice`	REAL(nmax)	sinc kernel index	`formatout`	CHARACTER	Format of output file
`indx`	INTEGER(nmax)	Particles IDs sorted by radius	`gamma`	REAL	Ideal gas EOS constant
`l`	INTEGER	Iteration counter	`gravity`	LOGICAL	Activate gravitational calculation
`masa`	REAL(nmax)	Particle mass	`inienergy`	LOGICAL	Initialize internal energy with EOS
`neighbors`	INTEGER(ALLOC)	List of neighbor particles IDs	`inivel0`	LOGICAL	Initialize velocities to 0
`npini,npend`	INTEGER	Initial and final particle ID/MPI	`inputdata`	CHARACTER	Input file
`nvi`	REAL(nmax)	Particle number or neighbors	`iodelay`	INTEGER	Iterations interval to write data
`omega`	REAL(nmax)	Grad-h term	`iterini`	INTEGER	Initial iteration
`p`	REAL(nmax)	Pressure	`kernel`	INTEGER	Selects kernel
`pk`	REAL(nmax)	Kernel normalization	`liniNR`	INTEGER	Iteration to start h update with Newton-Raphson
`pro`	REAL(nmax)	p/ρ^2	`maxtstep`	REAL	Max. allowed timestep
`promro`	REAL(nmax)	Density (ρ)	`n`	INTEGER	Number of particles
`rad`	REAL	Domain size	`ncubes`	INTEGER	Number of cubes in the PBC
`radius`	REAL(nmax)	Radial position of particles	`ngravproc`	INTEGER	MPI processes for gravity
`ro2`	REAL(nmax)	ρ^2	`nhydroproc`	INTEGER	MPI processes for hydrodynamics
`temp`	REAL(nmax)	Temperature	`ni0`	REAL	Target number of neighbors
`tt`	REAL	Physical time	`nmax`	INTEGER	Max. number of particles
`u`	REAL(nmax)	Internal energy	`nnl`	INTEGER	Max. number of iterations
`ugrav`	REAL(nmax)	Gravitational potential	`nodes`	INTEGER	Total number of MPI processes
`v`	REAL(dim$\times$nmax)	Velocity	`NRitermax`	INTEGER	Max. number of iteration when updating h
`vol`	REAL(nmax)	Volume elements (VE)	`nvmax,nvmin`	INTEGER	Limits to the number of neighbors
`volstdprom`	REAL(nmax)	SPH averaged VE	`outini`	LOGICAL	Write a dumpfile on the first iteration
`xbox,ybox,zbox`	REAL	Box size for PBC	`pexp`	REAL	Exponent for the VE
			`timeacc,...`	LOGICAL	Activate timestep control by magnitudes
			`timeini`	REAL	Initial physical time
			`timesave`	LOGICAL	Activate timestep control output
			`tol`	REAL	Opening parameter of the tree nodes for gravity