SEVN2
User guide

For any question/comment:
giuliano.iorio@unipd.it
giuliano.iorio.astro@gmail.com

Foreword...

- This document has been made with the idea to produce a presentation that can also be a reference manual for SEVN users. Therefore some (most) of the slides are quite verbose and not captivating as could be expected for an oral presentation. I am sorry for that, but I think you will appreciate it during your SEVN “session”.

- Preparing this document, I assumed a very general audience: from bachelor students with little coding/computing experience to skilled programmers and/or stellar evolution/ population synthesis experts. In order to be useful for the former, some of the contents could result trivial for the latter.

- The most updated version of this document and a number of examples of SEVN usage can be found at this link

Foreword...

- The most updated version of this document and a number of examples of SEVN usage can be found at this link

If you want to download these resources using wget, follow the steps:

```
wget https://www.dropbox.com/sh/h8xstil8giu0fjq/AADlK4YO5aslmxLIAxduXFt3a?dl=1 -O SEVN_tutorial.zip
```

```
unzip SEVN_tutorial.zip -d SEVN_tutorial
```

```
rm SEVN_tutorial.zip
```

The examples and the other resources can be found in the folder SEVN_tutorial

Summary

Brief introduction to SEVN

What is SEVN?

Stellar EVolution N-body
is a population synthesis code

- Written completely in C++ (C++14 standard, but compatible with older compiler)

- Stellar evolution through **interpolation of precomputed stellar tracks**

- Precomputed stellar tracks can be easily added to use the **most updated stellar evolution models**

- Binary processes through analytic and semi-analytic methods

- The evolution is performed with an **adaptive time step** schema giving more time resolution where needed

Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

Basic concepts:

- The SSE of a star in univocally defined by two values: **Ms**, **Zs**, i.e. the zero-age main sequence mass and its metallicity.

For each couple of (**Ms**, **Zs**) the SSE will be always the same (for a given set of evolutionary tracks).

Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

Basic concepts - stellar types/phases:

- The lifetime of a star in divided in different stellar phases:

Phase Int	Phase str	Physics	Hurley+02 Phase
0	PreMainSequence		
1	MainSequence	Core H-burning start	MS
2	TerminalMainSequence	Creation of a He core	HG/GB
3	HshellBurning	He core almost formed	GB
4	HecoreBurning	Core He-burning start	CHEB/HeMS*
5	TerminalHecoreBurning	Creation of a CO core	AGB/HeHG*/HeGB*
6	HeshellBurning	CO core almost formed	AGB
7	Remnant	Star is dead	WD/NS/BH

Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

Basic concepts - stellar types/phases:

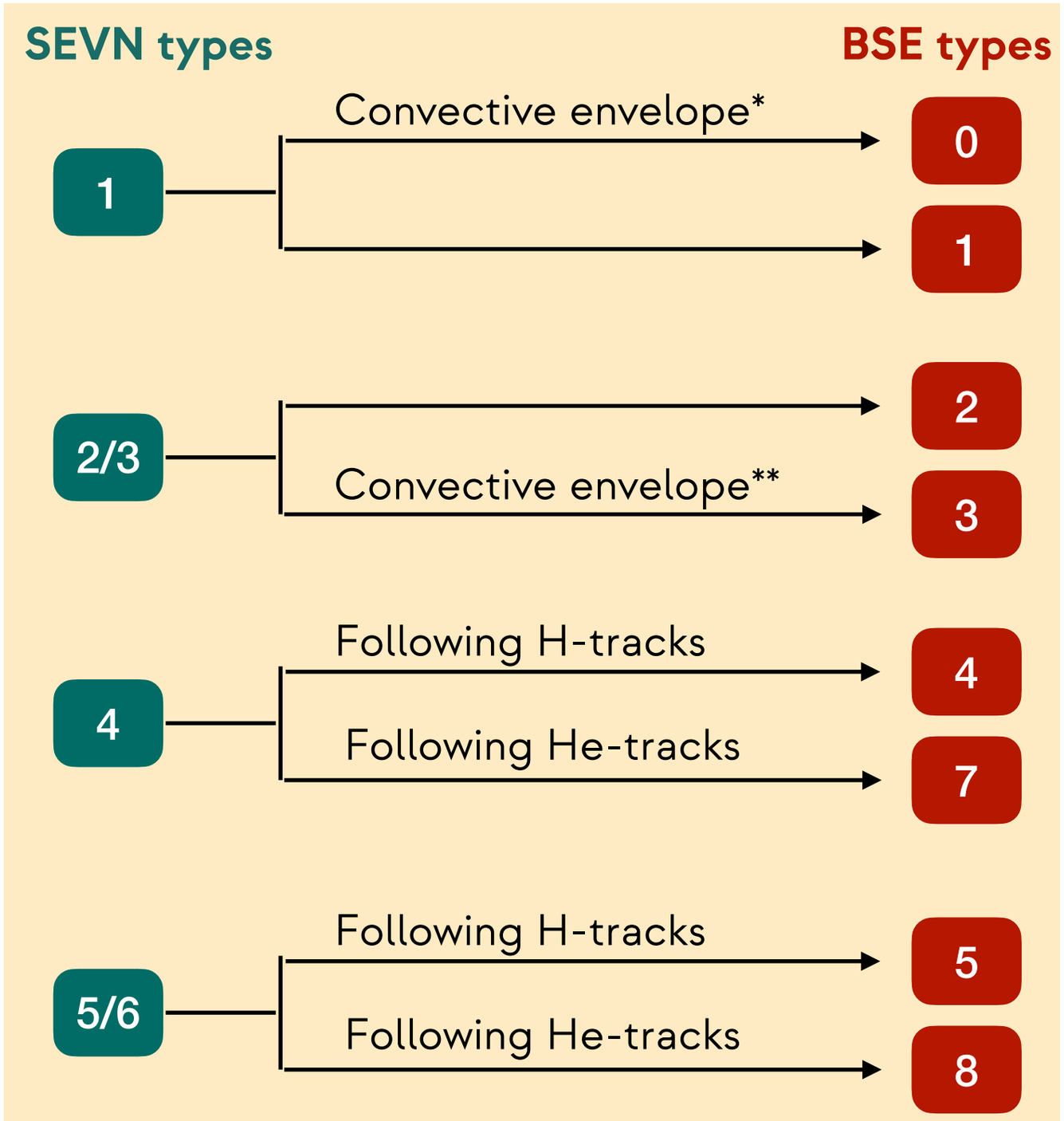
- The “after life” of a star in divided in different remnant types:

RemType Int	RemType str	Physics
0	NotARemnant	Star is not a remnant
1	HeWD	Helium White-Dwarf
2	COWD	Carbon-Oxygen White-Dwarf
3	ONeWD	Oxygen-Neon White-Dwarf
4	NS_ECSN	Neutron Star born after an Electro capture Supernova
5	NS_CCSN	Neutron Star born after a Core Collapse Supernova
6	BH	Black Hole
-1	Empty	Massless remnant

Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

Basic concepts - stellar types/phases: Used to BSE/MOBSE types?



*__Convective envelope__: if the table with fraction of convective envelope mass is available (Qsup), envelope is convective if Qsup>0.8, otherwise we use the condition Mzams<0.7 (see Hurley+02).

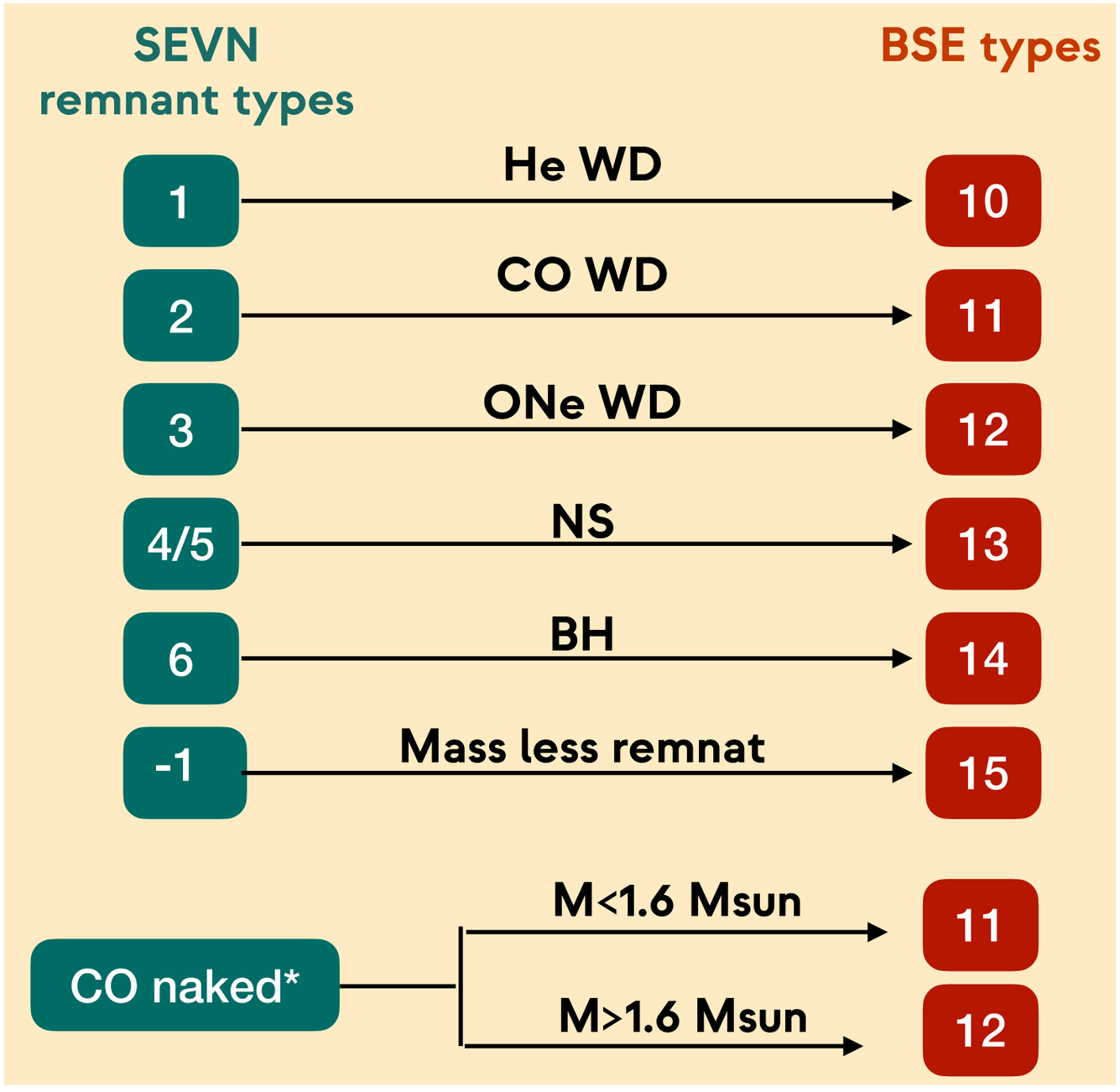
**__Convective envelope__: if mass of the convective envelope is larger than 33% of the mass of the envelope, the envelope is considered convective (see Hurley+00). If the table with the fraction of convective envelope mass is not available, Qsup is estimated following Hurley+00 (Sec. 7.2).

**__H/He tracks__: stars following H-tracks are star with an H-envelope and a He/CO core; stars following He-tracks have lost they H-envelope. See here for further details

Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

Basic concepts - stellar types/phases: Used to BSE/MOBSE types?



***CO naked**: during the BSE evolution is possible to turn a star into a bare CO core. These stars have not a special type, but they have a special treatment inside SEVN. When conversion to BSE types is needed they are considered as WD (the type of WD depend on the mass).

Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

Basic concepts:

- SEVN needs precomputed stellar tracks inserted in lookup tables.

Each stellar track refers to a star with zero-age main sequence mass Mzams and metallicity Z.
The lookup tables contain the stellar properties (e.g. Mass, Radius, Luminosity) at given times.

There are two kind of tables:

- **H-stars tables:** tracks containing the stellar evolution. of “standard” stars with Hydrogen envelopes.

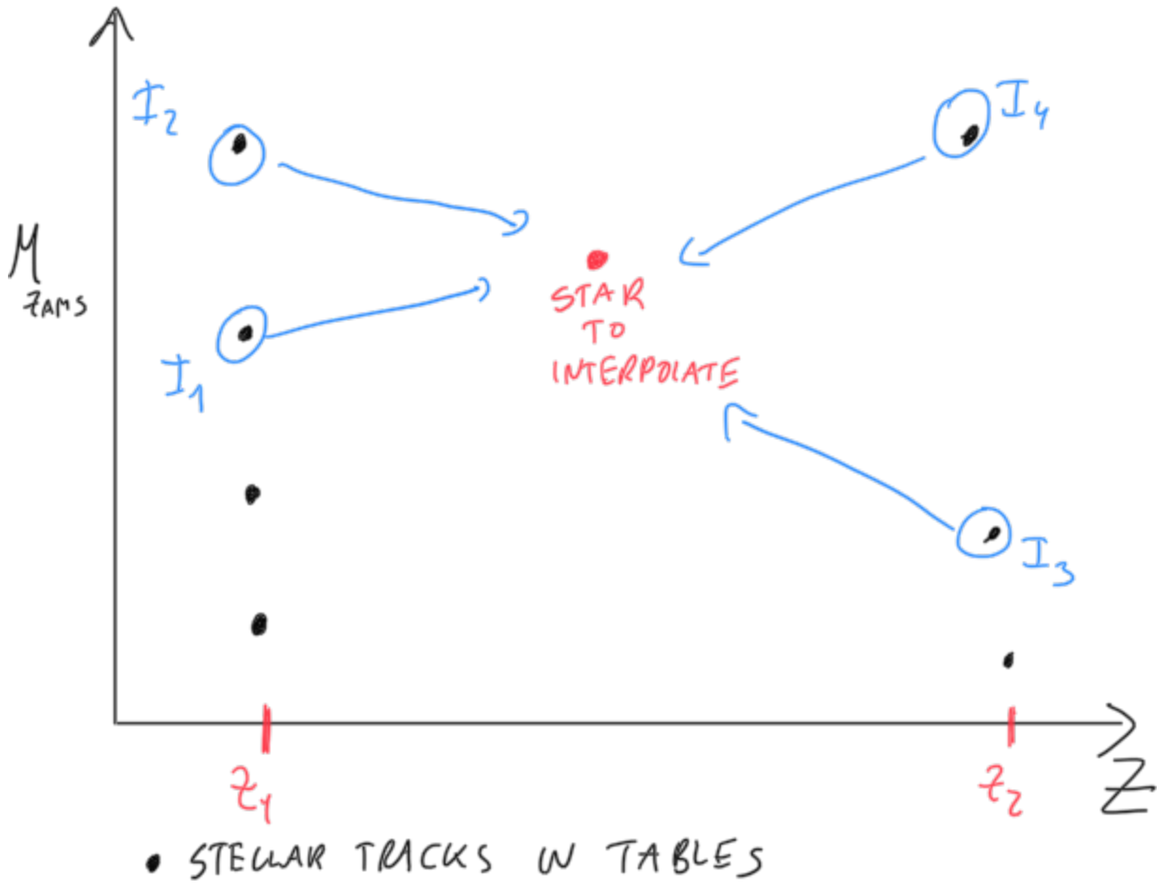
- **He-stars (pureHe) tables**: tracks containing the stellar evolution of pureHE stars without Hydrogen envelopes.

Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

Basic concepts:

- For each couple of (**Ms**, **Zs**) SEVN assigns **four interpolating tracks** taken from the tables: I1(M1<=Ms,Z1<=Zs), I2 (M2>=Ms,Z1<=Zs), I3 (M3<=Ms,Z2>=Zs), I4 (M4>=Ms,Z2>=Zs). Where the value of M* and Z* are the closest upper and lower (or equal) values wrt (**Ms**, **Zs**) found in the tables.



Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

Basic concepts:

- Properties are obtained from the interpolating tracks at the **same life percentage** in a given stellar phase:

$$\mathrm{plife} = \frac{\tilde{t} - t_{0,\mathrm{phase}}}{\Delta t_{\mathrm{phase}}}$$

Using plife instead of time we are guaranteed that all the interpolating tracks are in the same stellar phase of the interpolated star (i.e. they all have or not a HE/CO core).

Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

SSE Evolution - Some definitions:

Interpolating tracks: evolutionary tracks at given M and Z used to interpolate the properties and their evolution of a star. Their properties are stored in lookup tables. They are evolved using linear interpolation.

Fake star: it is evolved weighting the values of four interpolating tracks.

Real star: it is evolved using the relative difference of the fake star. This is star that we actually got as output.

Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

SSE Evolution - Step-by-step:

Estimate plife of the Real star

$$\text{plife} = \frac{\tilde{t}_s - t_{0,\text{phase},s}}{\Delta t_{\text{phase},s}}$$

Evolve the interpolating tracks

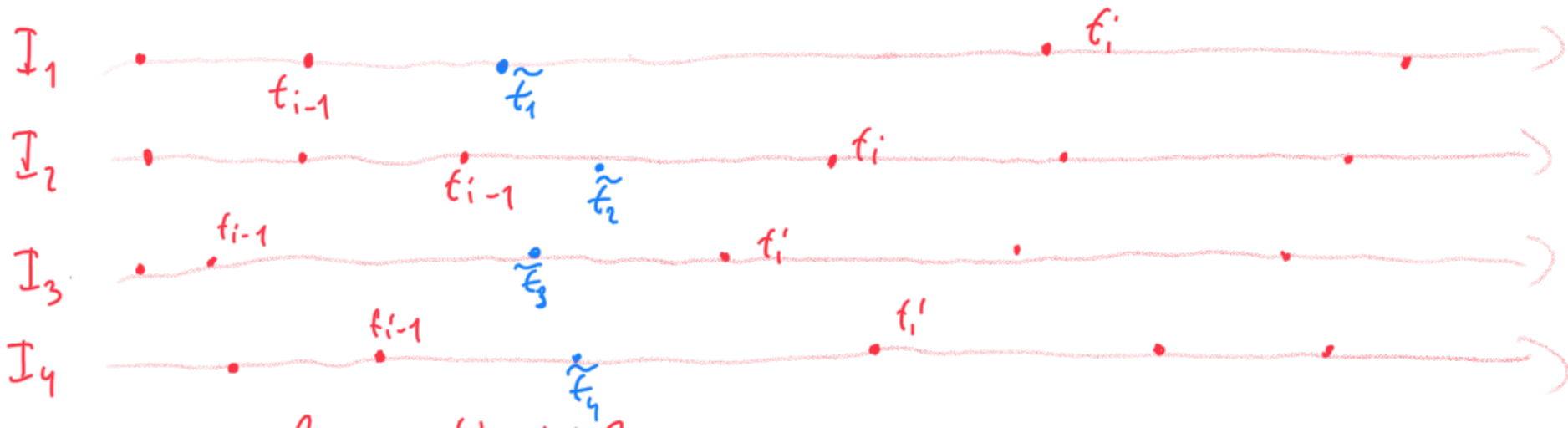
Evolve the Fake star

Evolve the Real star

Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

SSE Evolution - Step-by-step- Evolve the interpolating tracks:



$$P_{\tilde{t}} = \frac{P_{t_i} - P_{t_{i-1}}}{t_{i-1} - t_i} (\tilde{t} - t_{i-1})$$

linear interpolation

Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

SSE Evolution - Step-by-step- Evolve the Fake star:

Weighted combination from the interpolate tracks at the same plife.

$$Y_1 = \beta_1 P_{\mathrm{I}_1}(\tilde{t}_1) + \beta_2 P_{\mathrm{I}_2}(\tilde{t}_2) \qquad Y_2 = \beta_3 P_{\mathrm{I}_3}(\tilde{t}_3) + \beta_4 P_{\mathrm{I}_4}(\tilde{t}_4)$$

$$P_{\mathrm{fake}}(\tilde{t}_{\mathrm{s}}) = \gamma_1 Y_1 + \gamma_2 Y_2$$

$$\beta_{1/3}{}^{*} = \frac{M_{\mathrm{zams},2/4} - M_{\mathrm{s}}}{M_{\mathrm{zams},2/4} - M_{\mathrm{zams},1/3}} \qquad \beta_{2/4}{}^{*} = \frac{M_{\mathrm{s}} - M_{\mathrm{zams},1/3}}{M_{\mathrm{zams},2/4} - M_{\mathrm{zams},1/3}} \qquad \gamma_1 = \frac{Z_2 - Z_{\mathrm{s}}}{Z_2 - Z_1} \qquad \gamma_2 = \frac{Z_{\mathrm{s}} - Z_1}{Z_2 - Z_1}$$

* Some properties (e.g. Luminosity) uses logarithmic weights (based on log M).

Single Stellar Evolution (SSE)

SSE through **interpolation of precomputed stellar tracks**

SSE Evolution - Step-by-step- Evolve the Real star:

Evolve following the relative difference of the Fake star

$$P_{\rm real}(\tilde{t}_{\rm s}) = P_{\rm real,0} + \frac{P_{\rm fake}(\tilde{t}_{\rm s}) - P_{\rm fake,0}}{P_{\rm fake,0}} P_{\rm real,0}$$

Why not using directly the fake star?

The Fake and the Real stars are equivalent in case of SSE only. When BSE is included, properties (e.g. the Masses) can changes due to binary processes, thus the division between the fake and the real star is necessary to follow the variation due to both the SSE and the BSE.

Binary stellar evolution (BSE)

BSE through **analytics and semi-analytics methods**

- Wind Mass Transfer
- Roche-Lobe Overflow (stable/unstable)
- Common Envelope
- Stellar Merger/Collision
- Stellar tides
- Orbital decay due to Gravitational Wave emission
- SN explosion

- Implemented as analytic/semi-analytic simplification of complex physical processes
- They depend on parameters representing our knowledge/ ignorance
- **Choice of different parameters can completely change the fate of a binary system** (e.g. survive or not after a Common Envelope)

Change of track

During the BSE, a star may accrete or lose mass. When this happens SEVN looks for a new stellar track (i.e. a star with a new Mzams) that is more consistent with the current stellar properties, i.e. having a given mass at a given plife.

Change of track criteria:

- **During the MS, the star always jumps** if it accretes mass or lose mass (not considering stellar winds) more than a given threshold (see here).

- **Outside the MS**, when a core is present, **a star jumps only if it is forced by some process** (e.g. after a merger).

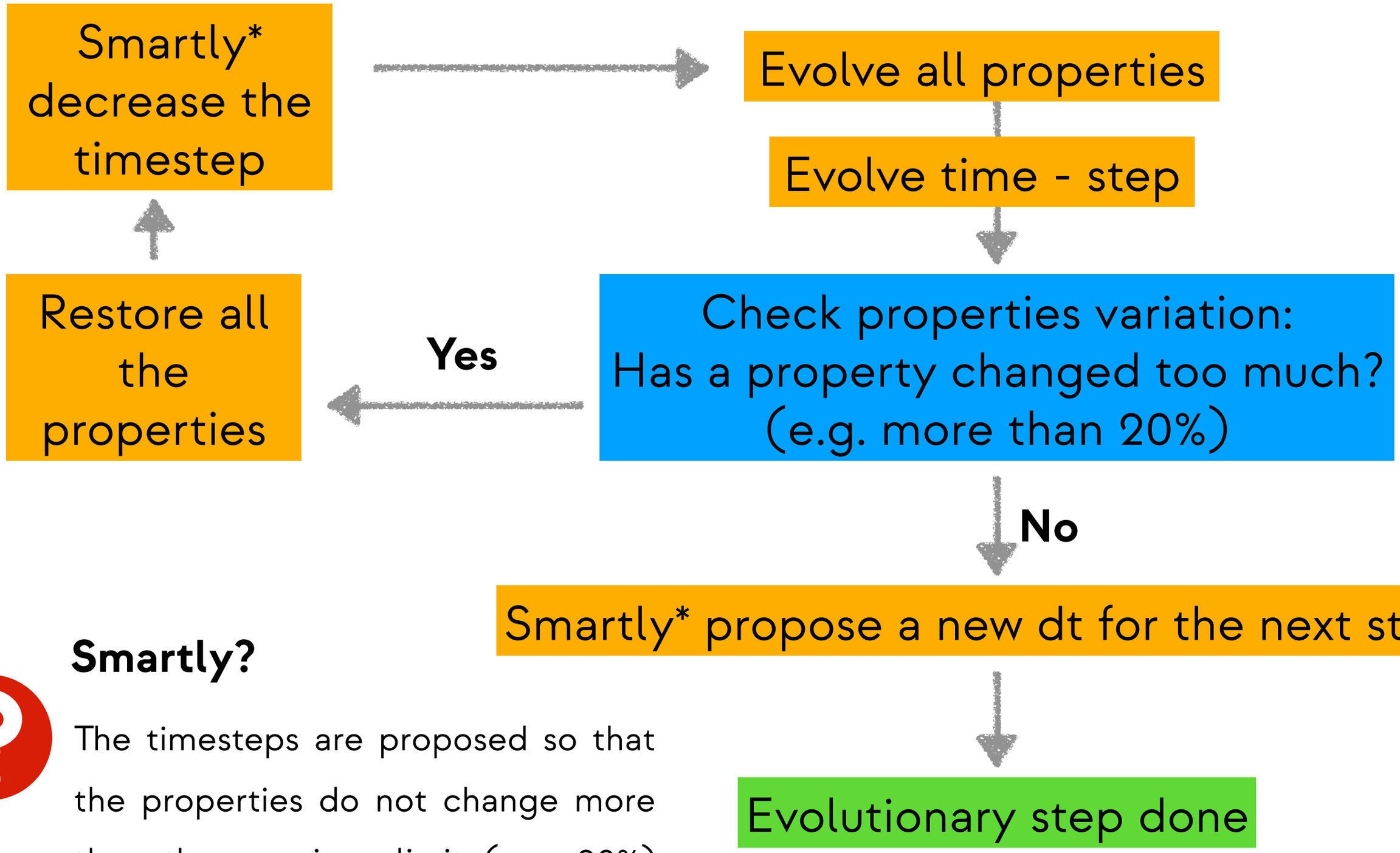
- **If a star loses its envelope it always jumps** to a new HE-track.

Change of track strategy:

- **A MS star** jumps to a new track trying to **match the total mass** at the same plife.

- **Outside the MS,** a star jumps to a new track **trying to match the total mass or the innermost core mass** at the same plife. The track with the best match of the Mass/Mcore ratio is chosen.

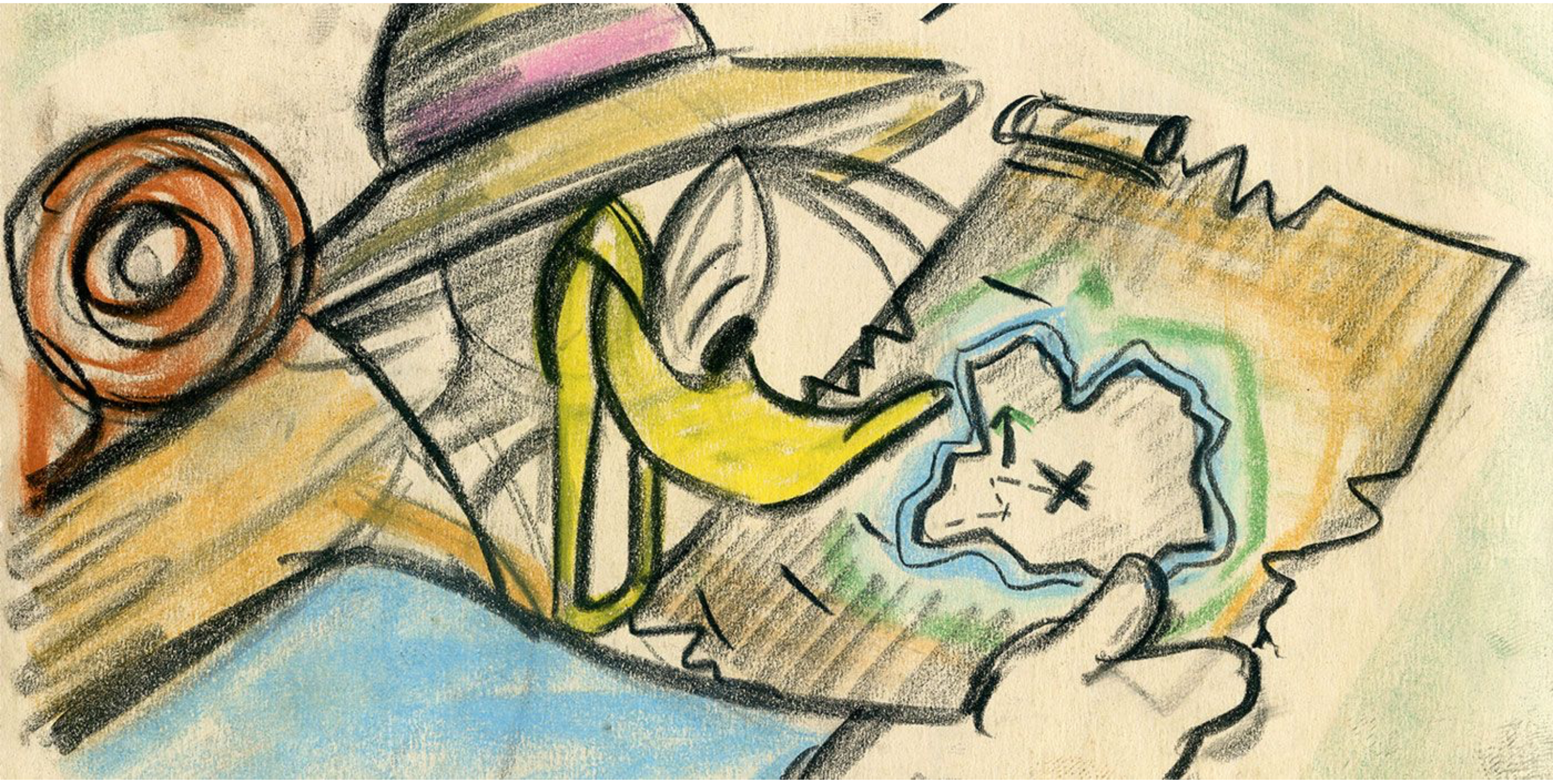
The adaptive time-step schema



Smartly?

The timesteps are proposed so that the properties do not change more than then a given limit (e.g. 20%) considering the last step differences and assuming a linear evolution.

Getting SEVN2



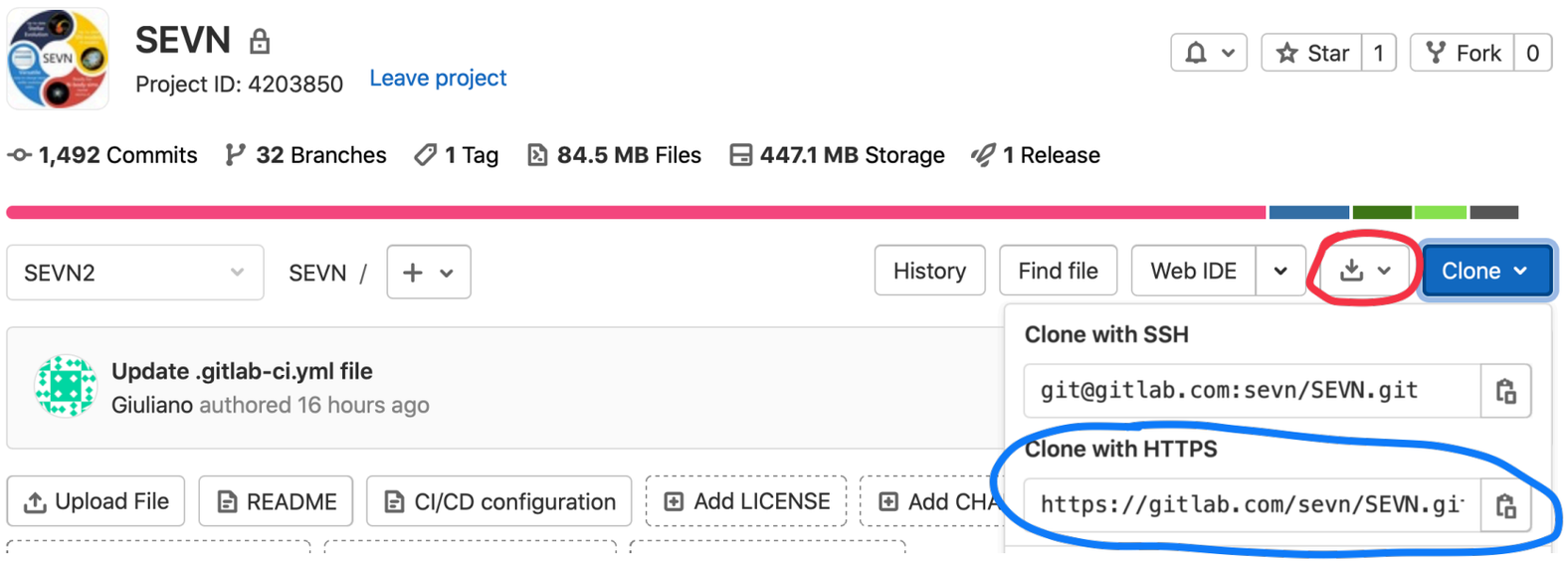
SEVN2 repository

SEVN2 is in a Gitlab repository:
https://gitlab.com/sevn/SEVN

The repository is private, an invitation to the project is needed. If you are interested, please contact us:

Giuliano Iorio: giuliano.iorio@unipd.it
Michela Mapelli: michela.mapelli@unipd.it
Mario Spera: mario.spera@live.it
Guglielmo Costa: guglielmo.costa@unipd.it

Getting SEVN2 from repository



Two methods to get SEVN2 from the repository:

1- **Get the source file**: use the red highlighted icons to download the source in a given archive format or use this link.

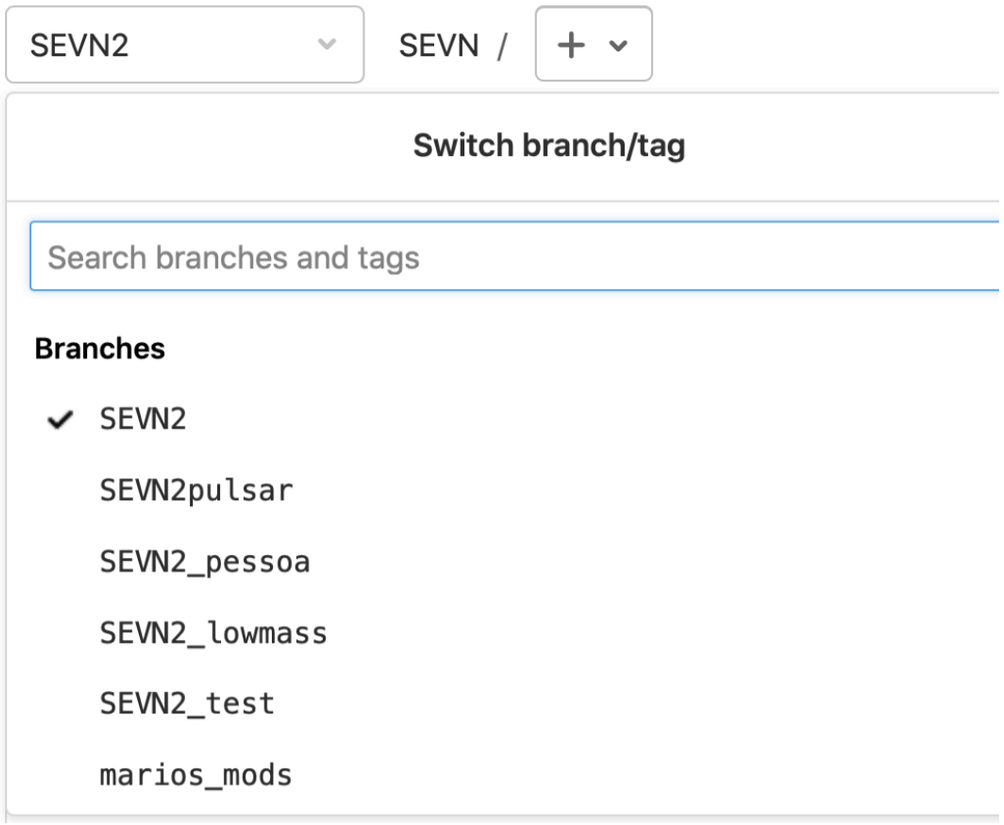
2- **Clone with git (git required, recommended)**: if you have git, it is possible to clone locally the repository using the blu highlighted link (NOT use SSH), i.e.

```
git clone https://gitlab.com/sevn/SEVN.git
```

The local repository can be updated with:

```
git pull
```

SEVN2 branches



The SEVN repository has many branches, (`git branch -a` to show all the branches or see this link). **The most updated stable branch** for which this tutorial is made **is SEVN2.** All the other branches contains experimental features or are not updated. **NOT USE THEM**.

SEVN2 structures

The SEVN2 folder contains many subfolders and files

Code folders:

include: contains the code headers (.h)
src: contains the code src file (.cpp)
extern/catch: contain the src code for unit test
auxiliary_data: data/tables used in the code (not stellar tables)
tables: contains the stellar tables used in SEVN to evolve stars
test: contains the src file to perform code testing
main: contains the src file to perform BSE and SSE evolution

Compilation:

CMakeList.txt: main instruction for Cmake
cmake: contains additional cmake module used in Cmake
compile.sh: user-friendly script to facilitate the code compilation

Utilities:

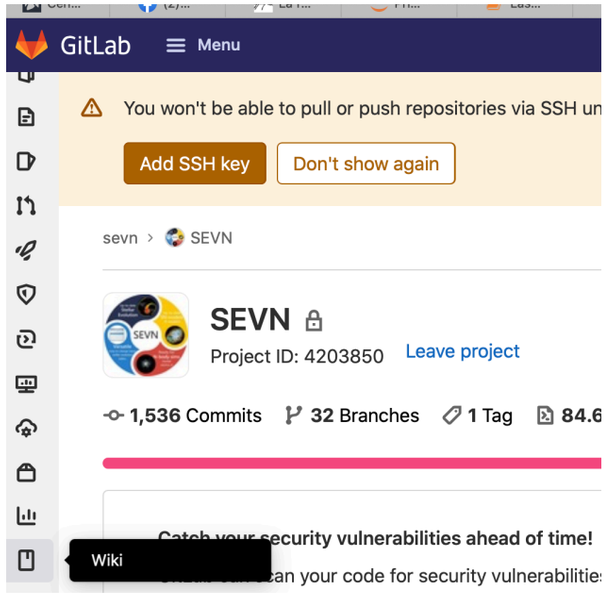
run_scripts: contains user-friendly script to facilitate the SEVN run
pyscript: contains a collection of useful python scripts
hdf5_interpreter: utilities to analyse hdf5 files

A standard SEVN2 user will use only the **compile.sh** and the **run scripts** (maybe the python scripts).

SEVN2 documentation and issue

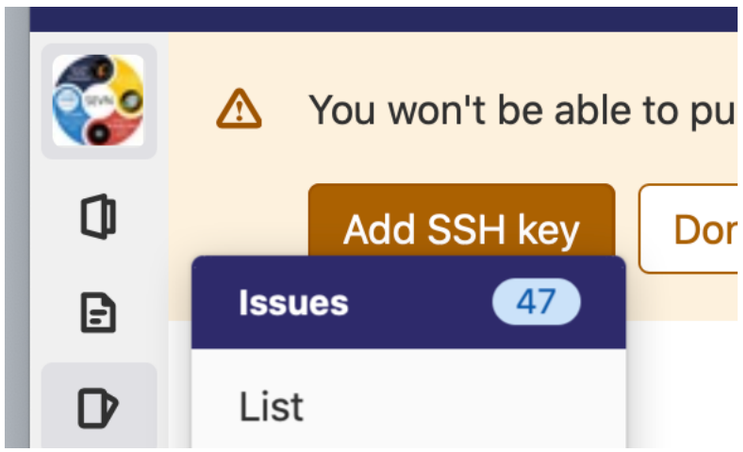
The git repository contains also:

Official Wiki/Documentation



The wiki (https://gitlab.com/sevn/SEVN/-/wikis/home) contains the most updated documentation about the code.

Issues



The issues page (https://gitlab.com/sevn/SEVN/-/issues) lists all the known issues/bugs of the code. If you find something "anomalous" or not consistent with the documentation while running SEVN, please open an issue here.

Compiling SEVN2

Requirements and compatibility

See this wiki page for more information

Requirements:

- **Cmake** (V>2.8, recommended V>3.2): SEVN2 uses Cmake for compilation.

- **C++ compiler**:

- **GNU C++:** SEVN2 has been tested with GNU compilers with version > 4, a version>7 is recommended.

- **Intel:** SEVN2 is compatible with the C++ compiler from the Intel one API, however it has not been extensively tested.

- **Clang:** SEVN2 cannot be compiled with Clang due to lack of openmp support.

Requirements and compatibility

See this wiki page for more information

Compatibility:

Linux

SEVN2 has been extensively tested on Linux machines.

The Cmake and C++ compiler can be get with the apt package manager:

- **Cmake**: *apt-get install cmake*
- **GNU C++:** *apt-get install g++*

Requirements and compatibility

See this wiki page for more information

Compatibility:

- SEVN2 has been extensively tested on macOS >10.14.
- SEVN2 should be compatible with macOS >10.7.
- It has never been tested on new macOS with the new M1 processors.

The default C++ compiler is Clang that is not compatible with SEVN2

A guide to satisfy SEVN2 requirements:

- **Command Line Tools**: this is required to enable all the other command, open a terminal and type *xcode-select —install*
- **Homebrew**: homebrew is a package manager for macOS and facilitate the installation of Cmake and the C++ compiler. To install homebrew follow the instruction at brew.sh
- **Cmake**: *brew install cmake* (brew link)
- **GNU C++**: *brew install gcc* (brew link)

Requirements and compatibility

See this wiki page for more information

Compatibility:

macOS

- SEVN2 has been extensively tested on macOS >10.14.
- SEVN2 should be compatible with macOS >10.7.
- It has never been tested on new macOS with the new M1 processors.

The default C++ compiler is Clang that is not compatible with SEVN2

A guide to satisfy SEVN2 requirements:

Notice, after the installation of the hombrew gcc, the default gcc and g++ compiler (the one in /usr/local/bin) will be still the clang ones, therefore the SEVN compilation will fail. In order to use the right version you can:

- **overwrite** the gcc and g++ in the /usr/local/bin
- **execute** the command *export CXX=g++-<version>* and *export CC=gcc-<version>*, where version is the c++ version installed with homebrew. Only the major version has to be considered, i.e. if the gcc version is 10.5.2, the command to run is *CXX=g++-10*. Important: these commands have to be run each time SEVN has to be compiled

In order to find the installed version and to find the executable to overwrite the standard one in /usa/local/bin, notice that the homebrew gcc can be fond in **/usr/local/Cellar/gcc**

Requirements and compatibility

See this wiki page for more information

Compatibility:

HIC SUNT LEONES!

Not tested, not known!

Requirements and compatibility

See this wiki page for more information

Alternative solution: CONDA®

Sometime is not possible to updated packages or install them directly.
A possible solution is to exploit the CONDA package manager (working on all operative systems).
The easiest way to get CONDA is to install ANACONDA.
Once Conda is installed:

- Download the environment file: conda_sevn_env.txt
  Notice: this environment is only for a linux system

- Creat the conda environment with
  `conda create --name <env_name> --file conda_sevn_env.txt`

- Activate the conda environment with
  `conda activate <env_name>`

- Compile and use SEVN2 as described in this tutorial

Compiling

Two options:

1. Manually use CMake

2. Use an automated script we kindly provided

Compiling

CMake: See this wiki page for more information

1. Open the terminal and go to the SEVN folder
2. Remove the *build* folder, if present, and create a new one

```
rm -rf build; mkdir build
```

3. Enter in the build folder and run cmake

```
cd build; cmake ..
```

4. If everything run smooth, you should see the following message

```
--Configuring done
--Generating done
--Build files have been written to: <current build path>
```

5. Compile with Make

```
make -j
```

6. If everything run smooth, you should see the following message

```
Linking CXX executable sevn.x
Built target sevn.x
Linking CXX executable sevnB.x
Built target sevnB.x
```

Compiling

Installation script: See this wiki page for more information

1. The installation script *compile.sh* is inside the SEVN folder
2. Make the script executable

```
chmod u+x compiles.sh
```

3. Run it (from anywhere, the script does not need to remain in the SEVN folder)

```
./compile.sh -p <PATH_TO_SENV_FOLDER>
```

3b. Alternatively, you can insert the path to SEVN folder directly in compile.sh file at the line SEVN="<Insert absolute SEVNpath>", in that case just use

```
./compile.sh
```

4. If everything run smooth you should see this message

```
**********************
SEVN has been successfully compiled
…
```

This script has other options, call ./compile.sh -h to list them

Compiling - only make

See example 0

CMake is used to “make” the Makefile, then the Makefile is used to compile SEVN. **Therefore to recompile SEVN it is not needed to call cmake again.**

CMake:

1. Go inside the SEVN folder and type

```
make -j
```

Installation script:

1. Run the installation script with the the option *-m*

```
./compile.sh -p <PATH_TO_SENV_FOLDER> -m
```

1b. If the path to the SEVN folder is written inside the script,

```
./compile.sh -m
```

Compiling - extra options

Cmake allows a number of option:

```
cmake .. [-Dbin=ON*/OFF] [-Dsin=ON*/OFF] [-Dh5=ON/OFF*] [-Ddebug=ON/OFF*] [-Dtest=ON/OFF*] [-Ddoc=ON/OFF*]
```

* indicates the default value

Standard users

-**Dbin** Enable the compilation of the BSE executable *sevnB.x*

-**Dsin** Enable the compilation of the SSE only executable *sevnB.x*

-**Dh5** Enable the hdf5 compilation for output

Developers

-**Ddebug** Enable the debug run mode, notice this can be set at runtime

-**Dtest** Compile the test suit, only possibile for CMake V>3.2

-**Ddoc** Produce the doxygen documentation

The *compile.sh* script always uses the default cmake options.

Running SEVN2

Executables

After a successful compilation the executables will be found in the folder

SEVN/build/exe

sevnB.x

Executable for the SSE+BSE evolution. Input systems are binaries

sevn.x

Executable for the SSE evolution only. Input systems are single stars

Both are executed in the same way and accept the same parameters, the only difference is the input list format.
In the following slides we will use just *sevnB.x*.

How to run

Two options:

1. Manually run the executable

```
./sevnB.x –list <path_to_input_list> [optional params]
```

list is the only required parameters, all the other have default value and can be omitted.

Notice, in this example we assume that we are in the same folder of the *sevnB.x* executable. It his is not the case use

```
<path_to_the_executable_directory>/./sevnB.x
–list <path_to_input_list> [optional params]
```

How to run

See example 1

Two options:

2. Use automated scripts we kindly provided

   The automated scripts are in the folder *SEVN/run_scripts/*

   1. Make the script executable

      ```
      chmod u+x run.sh
      ```

   2. Open the *run.sh* script with a text editor

   3. Set the variable SEVN as the path to the SEVN folder

      ```
      SEVN=<Insert_absolute_sevn_path>
      ```

   4. Set other optional variables. **Notice all the options are already set in the file with their default value**

   5. Run the script, inserting as argument the path to the input list

      ```
      ./run.sh <path_to_input_list>
      ```

You don't have to run the script in the folder you find it (SEVN/run_scripts/), the idea is that can be moved anyway, while the SEVN executables remain in the SEVN folder

How to run - from download to run

A complete walkthrough

1- We start downloading (cloning) the code from gitlab

```
(base) iogiul@Giulianos-MacBook-Pro SEVN_tutorial % git clone https://gitlab.com/sevn/SEVN.git
Cloning into 'SEVN'...
remote: Enumerating objects: 29131, done.
remote: Counting objects: 100% (632/632), done.
remote: Compressing objects: 100% (277/277), done.
remote: Total 29131 (delta 373), reused 613 (delta 355), pack-reused 28499
Receiving objects: 100% (29131/29131), 87.67 MiB | 3.89 MiB/s, done.
Resolving deltas: 100% (22139/22139), done.
Updating files: 100% (2959/2959), done.
```

Git clone

```
(base) iogiul@Giulianos-MacBook-Pro SEVN_tutorial % ls
SEVN
(base) iogiul@Giulianos-MacBook-Pro SEVN_tutorial % cd SEVN
(base) iogiul@Giulianos-MacBook-Pro SEVN % ls
CMakeLists.txt   cmake            extern           listBin.dat      pyscript         tables
README.txt       compile.sh       hdf5_interpreter listSin.txt      run_scripts      test
auxiliary_data   doc              include          main             src
```

The SEVN folder

```
(base) iogiul@Giulianos-MacBook-Pro SEVN % git branch
* SEVN2
(base) iogiul@Giulianos-MacBook-Pro SEVN %
```

Check the branch

How to run - from download to run

A complete walkthrough

2- Locate the important files

```
(base) iogiul@Giulianos-MacBook-Pro SEVN % ls
CMakeLists.txt   cmake            extern            listBin.dat      pyscript         tables
README.txt       compile.sh       hdf5_interpreter  listSin.txt      run_scripts      test
auxiliary_data   doc              include           main             src
```

```
(base) iogiul@Giulianos-MacBook-Pro SEVN % cd run_scripts
(base) iogiul@Giulianos-MacBook-Pro run_scripts % ls
listBin.dat  listStar.dat  run.sh       run_sse.sh
```

compile.sh: script used to compile SEVN

run.sh: script used to run SEVN (BSE version)

listBin.dat: template to be used as input file (it contains only 1 system)

How to run - from download to run

A complete walkthrough

3- Create a “run” folder and copy the script files

Notice: the “run” folder can be created everywhere and it do not need a specific name.

```
(base) iogiul@Giulianos-MacBook-Pro SEVN % cd ..
(base) iogiul@Giulianos-MacBook-Pro SEVN_tutorial % mkdir SEVN_run
(base) iogiul@Giulianos-MacBook-Pro SEVN_tutorial % ls
SEVN     SEVN_run
(base) iogiul@Giulianos-MacBook-Pro SEVN_tutorial % cd SEVN_run
```

Create folder

```
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % cp ../SEVN/{compile.sh,run_scripts/run.sh,run_scripts/listBin.dat} .
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % ls
compile.sh  listBin.dat run.sh
(base) iogiul@Giulianos-MacBook-Pro SEVN_run %
```

copy files

Reasons behind the compilation and run scripts:

The standard and suggested SEVN2 usage assumes that for each run a dedicated folder is created and the compilation and run script are used.

The reason behind this guideline is to leave the SEVN folder as clean as possible, moreover, the executables are not moved around.

The run script directly contains the SEVN runtime parameters, so that executables and parameters are bound together improving the reproducibility and reducing the risk of confusing the set of parameters related to a given output.

How to run - from download to run

A complete walkthrough

3- Compile SEVN

open the compile.sh file with your favourite editor

```
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % vi compile.sh
```

locate the variable SEVN

```
#!/usr/bin/env bash

SEVN="<Insert absolute SEVNpath>" #Complete path to the SEVN folder
```

Set the variable SEVN with the path of the SEVN folder

```
#!/usr/bin/env bash

SEVN="/Users/iogiul/SEVN_tutorial/SEVN" #Complete path to the SEVN folder
```

How to run - from download to run

A complete walkthrough

3- Compile SEVN

Make the script executable

```
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % ls
compile.sh  listBin.dat run.sh
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % chmod u+x compile.sh
```

run it

```
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % ./compile.sh

-- Generating Makefile for the SEVN code
CMake Deprecation Warning at CMakeLists.txt:1 (cmake_minimum_required):
  Compatibility with CMake < 2.8.12 will be removed from a future version of
  CMake.
```

If SEVN has been successfully compiled, this message appears

```
**************************************************************************************************
SEVN has been successfully compiled
We recommend to set all the relevant parameters in one the run script inside the run_scripts folder and to run
SEVN through the same script (e.g. ./run.sh).
Have fun!
**************************************************************************************************
```

How to run - from download to run

A complete walkthrough

4- Prepare the run script

open the run.sh file with your favourite editor

```
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % ls
compile.sh  listBin.dat run.sh
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % vi run.sh
```

locate the variable SEVN

```
#-------------------------------
#SEVNPATH
#-------------------------------
SEVN="<Insert absolute SEVNpath>" #Complete path to the SEVN folder
```

Set the variable SEVN with the path of the SEVN folder

```
#-------------------------------
#SEVNPATH
#-------------------------------
SEVN="/Users/iogiul/SEVN_tutorial/SEVN" #Complete path to the SEVN folder
```

How to run - from download to run

A complete walkthrough

4- Prepare the run script

Locate the variable LISTBIN

```
#------------------------------
#Input and Tables
#------------------------------
if [ "$cmdargument" = "" ]; then
  LISTBIN="${SEVN}/run_scripts/listBin.dat" #Complete path to input file (list of bi
 -
```

Set the variable LISTBIN with the path to the input file
(in our case listBin.dat)

```
#------------------------------
#Input and Tables
#------------------------------
if [ "$cmdargument" = "" ]; then
  LISTBIN="listBin.dat" #Complete path to input file (list of binaries or si
 -
```

Notice, in this example we assume that the input file is the file *listBin.dat* in the same directory of the run script. If this not the case, use the right path to set LISTBIN

How to run - from download to run

A complete walkthrough

5- run the script

Make the script executable

```
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % chmod u+x run.sh
(base) iogiul@Giulianos-MacBook-Pro SEVN_run %
```

Run the script

```
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % ./run.sh
```

If SEVN run successfully, this message appears

```
SEVN execution was SUCCESSFUL!
The output can be found in sevn_output/
```

How to run - from download to run

A complete walkthrough

6- check the output

The output are inside the sevn_output folder

```
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % ls
compile.sh  listBin.dat run.sh      sevn_output
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % cd sevn_output
(base) iogiul@Giulianos-MacBook-Pro sevn_output % ls
evolved_0.dat     launch_line.txt   output_0.csv      used_params.svpar
(base) iogiul@Giulianos-MacBook-Pro sevn_output %
```

Congratulations! You run your first SEVN simulation

How to run - from download to run

A complete walkthrough

Change the runtime options:

Suppose we want to run SEVN changing some of its default parameters,
we can due it directly in the script. E.g. suppose we want to use 2 threads in the run and change the name of the output folder

Locate the name of the parameter you want to change

```
#------------------------------
#RUN OPTIONS
#------------------------------
NTHREADS="1" #Number of OpenMP threads
```

```
#------------------------------
#OUTPUT
#------------------------------
OUTPATH="sevn_output/" #Complete path to the
```

Set new parameter value

```
#------------------------------
#RUN OPTIONS
#------------------------------
NTHREADS="2" #Number of OpenMP tl
```

```
#------------------------------
#OUTPUT
#------------------------------
OUTPATH="my_output" #Complete path
```

Run again the SEVN script

```
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % ./run.sh
SEVN execution was SUCCESSFUL!
The output can be found in my_output
(base) iogiul@Giulianos-MacBook-Pro SEVN_run % ls
compile.sh  listBin.dat my_output   run.sh      sevn_output
```

Input file

The single or binary systems to simulate has to be listed in an ascii file.
Each row must contain a single binary system or a star.

Input order:

Single star  See example 2bis 

M	Z	Ω	sn	$t_{\rm start}$	$t_{\rm end}$	$dt_{\rm out}$	$seed$ [optional]

Binary  See example 2 

M_1	Z_1	Ω_1	sn_1	$t_{\rm start,1}$	M_2	Z_2	Ω_2	sn_2	$t_{\rm start,2}$	a	e	t_{end}	$dt_{\rm out}$	$seed$ [optional]

1 is for the primary star, 2 for the secondary

Wiki page about SEVN input at this link:
https://gitlab.com/sevn/SEVN/-/wikis/SEVN-V2/run-the-code#input

Input file

See example 2

Binary

M_1	Z_1	Ω_1	sn_1	$t_{\text{start},1}$	M_2	Z_2	Ω_2	sn_2	$t_{\text{start},2}$	a	e	t_{end}	dt_{out}	$seed$ [optional]

1 is for the primary star, 2 for the secondary

M - Mass [Msun]

Set the stellar mass (and type).

- **Standard:** double value with the ZAMS mass in Msun, e.g. *30.5*. A Hydrogen star will be initialised

- **PureHE stars:** double value with the zero-age-helium-main-sequence mass in Msun + HE suffix, e.g. *30.5HE*. A pureHE stars will be initialised.

- **Remnants:** double value with the remnant mass + remnant-label suffix:

  - BH, e.g. *30.5BH*. A black-hole of a given mass will be initialised.

  - NS, e.g. 1.*5NS*. A of a neutron star of given mass will be initialised.

  - HEWD, COWD, ONEWD, e.g. *0.9HEWD*. A white dwarf (Helium WD, Carbon-Oxygen WD, Oxygen-Neon WD) of a given mass will be initialised.

If the remnant mass is not compatible with the remnant type (e.g. M> Chandasekar mass for WDs), SEVN returns an initialisation error.

Input file

See example 2

Binary

M_1	Z_1	Ω_1	sn_1	$t_{\text{start},1}$	M_2	Z_2	Ω_2	sn_2	$t_{\text{start},2}$	a	e	t_{end}	dt_{out}	$seed$ [optional]

1 is for the primary star, 2 for the secondary

Z - Metallicity [abundance of Z elements]

Set the stellar metallicity

Ω - *Stellar Spin* [angular velocity over critical angular velocity]

Set the stellar spin as the fraction of angular velocity over critical angular velocity

$$\Omega = \frac{\Omega_s}{\Omega_c} \qquad \Omega_c = \sqrt{\frac{GM}{R_{eq}^3 \text{ (equatorial radius)}}}$$

sn - Supernova type [string]

Set the formalism to handle the transition to stellar remnant for massive stars, see here for further details

- **rapid,** rapid formalism from Fryer et al. 2012
- **delayed,** delayed formalism from Fryer et al. 2012
- **compactness,** compactness formalism from Mapelli et al. 2020

Input file

See example 2

Binary

M_1	Z_1	Ω_1	sn_1	$t_{\text{start},1}$	M_2	Z_2	Ω_2	sn_2	$t_{\text{start},2}$	a	e	t_{end}	dt_{out}	$seed$ [optional]

1 is for the primary star, 2 for the secondary

t_{start} - Initial stellar age [Myr or string]

Set the age of the star at the beginning of the simulation:

- **Standard:** double value with the age in Myr, e.g. *0.5.*

- **Phase:** str with the name of the phase. The star will be initialised at the beginning of the phase (see table below), e.g. *tms.*

- **Fraction of phase:** str with % + a life percentage + : + code of the phase (integer, see table below), e.g. *%30:2*, i.e. 30% of the life during the phase 2 (*tms).*

Name of the phases

- *zams*, 1 (Zero Age Main Sequence)
- *tms*, 2 (Terminal Main Sequence)
- *hsb*, 3 (H-shell Burning)
- *cheb*, 4 (Core-He Burning)
- *tcheb*,5 (terminal Core-He Burning)
- *hesb*, 6 (He-shell Burning)

Input file

See example 2

Binary

M_1	Z_1	Ω_1	sn_1	$t_{\text{start},1}$	M_2	Z_2	Ω_2	sn_2	$t_{\text{start},2}$	a	e	t_{end}	dt_{out}	$seed$ [optional]

1 is for the primary star, 2 for the secondary

***a* - Semimajor axis** [Rsun]

Set the initial separation of the stars in the binary system.

Double value in Rsun

Using a very large value (e.g. 1e30), the binary processes will be de-facto disabled.

***e* - Eccentricy**

Set the initial eccentricity of the stars in the binary system.

Double value between 0 (inclusive) and 1 (exclusive)

Input file

See example 2

Binary

M_1	Z_1	Ω_1	sn_1	$t_{\text{start},1}$	M_2	Z_2	Ω_2	sn_2	$t_{\text{start},2}$	a	e	t_{end}	dt_{out}	$seed$ [optional]

1 is for the primary star, 2 for the secondary

***t_{end}* - End time of the simulation** [Myr or string]
Set the age at which the simulation is stopped

- **Standard:** double value with the age in Myr, e.g. *10000.*
- ***end*:** str end, the simulation is stopped when both stars become remnants e.g. *end.*
- ***broken*:** str broken, the simulation is stopped when the binary system breaks (because of a SN explosion, or destruction of a star), e.g. *broken.*

Input file

See example 2

Binary

M_1	Z_1	Ω_1	sn_1	$t_{\text{start},1}$	M_2	Z_2	Ω_2	sn_2	$t_{\text{start},2}$	a	e	t_{end}	dt_{out}	$seed$ [optional]

1 is for the primary star, 2 for the secondary

dt_{out} - time schedule for output [Myr or string]

Set the times where the output will be stored

- **Standard:** double value in Myr, the outputs will be printed each dt, e.g. *0.1.*

- **List:** list of comma separated values (in Myr) within curly braces, the output will be printed exactly at the listed times, e.g. *{0.1,0.5,1,2,5}*

- **Interval:** three numbers (in Myr) separated by semicolons, the first number is the starting time for the output, the second number is the end time and the third number is the time interval, e.g. *1:2:0.1,* print output from 1 Myr to 2 Myr each 0.1 Myr.

- ***all:*** str all, print outputs at each timestep, e.g. *all*. Notice the number and times of the output depends on the **adaptive timestep**.

- ***end:*** str end, print just the first and last timestep, e.g. *end.*

- ***events:*** print the outputs only when an event is triggered (**see events list in the wiki**)

Input file

Binary

M_1	Z_1	Ω_1	sn_1	$t_{\text{start},1}$	M_2	Z_2	Ω_2	sn_2	$t_{\text{start},2}$	a	e	t_{end}	dt_{out}	$seed$ [optional]

1 is for the primary star, 2 for the secondary

***seed* - random seed number** [integer] See example 3

Set the random seed

The integer value is used as random seed, for all the stochastic processes of the evolution (e.g. SN kicks). A star with the same initial parameters (same SEVN parameters also) and the same seed will always experience the same evolution.

This parameter is useful for the reproducibility but also to reduce the stochasticity in the parameter exploration (e.g. analysing the effect of the metallicity)

This is an optional input, if you want to use it, it has to be used for all the binary systems/stars in the input file.
If it is used, the run-time parameter *rseed* (see later) must be set to true.

Input file - placeholders

See example 4

All the input parameters, except for the Mass(es), spin, semimajor axis, eccentricity and seed (if present) can be replaced in the input file by the **placeholder string**: ***xxx***

e.g.
11.8 xxx 0 xxx xxx 9.6 xxx 0 xxx 350 0.3 xxx xxx
a system with M1=11.8 Msun, M2=9.6 Msun, Spin1=Spin2=0, Sem. axis=350 Rsun, ecc=0.3 and placeholders for all the others parameters.

A parameter with a placeholder **must be set as a SEVN runtime option**: -Z, -snmode, -tini, -tf, -dtout (see this slide)

Placeholders are useful to create initial-conditions “templates”, where a population can be run in SEVN testing, for example, different metallicities of sn models without creating a new file.

Input file

Last remarks

- Binary systems and single stars cannot be combined in the same file.

- *sevnB.x* accepts only input files with binary systems, while *sevn.x* accepts only input files with single stars.

- An empty row interrupts the file reading, everything below this row will not be considered by SEVN.

- The numbers have to be written using decimal or exponential notation (e.g. 1e2).

- Rows with typos and/or not consistent values raise a failed_initialisation error in SEVN. This error will not halt the SEVN execution. All the initial conditions with problems will be reported in the failed_initialisation file in the outputs of the simulation. See example 5

Wiki page about SEVN input at this link:
https://gitlab.com/sevn/SEVN/-/wikis/SEVN-V2/run-the-code#input

Runtime options

The complete list of runtime options can be found here

Sevn2 accepts a long list of optional parameters, the following slides describe some of them, the ones that a user is more likely to interface with.

For each parameter we report both the name of the argument to be used if SEVN is run directly and the name of the variable in the run script referring to that parameter, e.g.

-list arg LISTBIN="arg"

Runtime options - input

The complete list of runtime options can be found here

List

-list arg LISTBIN="arg"

arg: String containing the path to the input file

default: no default, the argument is required

Both relative and absolute paths are ok

If the run script is used, the path to the list can be given as argument

```
./run.sh <path_to_input_list>
```

Random seed → See example 3 ←

-rseed arg RSEED="arg"

arg: String containing true or false

default: false

If true the input file has to contain the random seeds in the last column .
If the input file contains this last column, this option has to be set to true. See here.

Runtime options - input

See example 4

The complete list of runtime options can be found here

Overwrite list inputs

The following options can be used to overwrite some input values of the systems list:

Z > overwrite **Z - Metallicity** -Z arg Z="arg"

snmode > overwrite **sn - Supernova type** -snmode arg SUPERNOVA="arg"

tini > overwrite **tstart - Initial stellar age** -tini arg TSTART="arg"

tf > overwrite **tend - End time of the simulation** -tf arg TEND="arg"

dtout > overwrite **dtout - time schedule for output** -dtout arg DTOUT="arg"

arg: 'list' or same input type of the **systems list**
default: 'list', if 'list' the value will be taken from the **systems list**

If an option is not set to 'list' the parameter will be **overwritten for all the systems in the input list.**

If placeholders are used in the input list, **the option 'list' cannot be used** to set the correspondent parameter.

Runtime options - tables

The complete list of runtime options can be found here

Tables

H-star stellar tracks: -tables arg TABLES="arg"

He-star stellar tracks: -tables arg TABLESHE="arg"

arg: String containing the path to the folder containing the lookup tables

default:

H-stars: *<SEVN_folder>/tables/SEVNtracks_parsec_AGB*
He-stars: *<SEVN_folder>/tables/SEVNtracks_parsec_pureHe36*

The tables for H-stars and He-stars are the core of the stellar evolution, they are interpolated on-the-fly by SEVN.
H-star stellar tracks are used to evolve "standard" star with Hydrogen envelopes. He-star stellar tracks are used to evolve pureHE stars without Hydrogen envelopes.

See this wiki page for learn more about the tables included in SEVN.

Runtime options - tables

The complete list of runtime options can be found here

Tables in SEVN

H-stars:

Name	Mass range [Msun]	Z range	Notes
SEVNtracks_parsec_AGB	2.2 - 600	0.0001 - 0.04	Most updated tracks from PARSEC
SEVNtracks_Spera19	2.0 - 157	0.0001 - 0.06	PARSEC tracks used in Spera+19
SEVNtracks_MIST_AGB_gold	0.65 - 150	0.000014 - 0.045	MIST tracks (made with MESA)

He-stars:

Name	Mass range [Msun]	Z range	Notes
SEVNtracks_parsec_pureHE36	0.360 - 350	0.0001 - 0.05	Most updated pureHE tracks from PARSEC
SEVNtracks_parsec_pureHE	0.380 - 120	0.0001 - 0.05	pureHE tracks with alternative mass loss (Sanders+19)

See this wiki page for learn more about the tables included in SEVN.

Runtime options - run options

The complete list of runtime options can be found here

Nthreads → See example 6 ←

-nthreads arg NTHREADS="arg"

arg: integer containing the number of parallel threads to use
default: 1

Nchunk

-ev_Nchunk arg NCHUNK="arg"

arg: integer containing the number of systems to evolve at each step

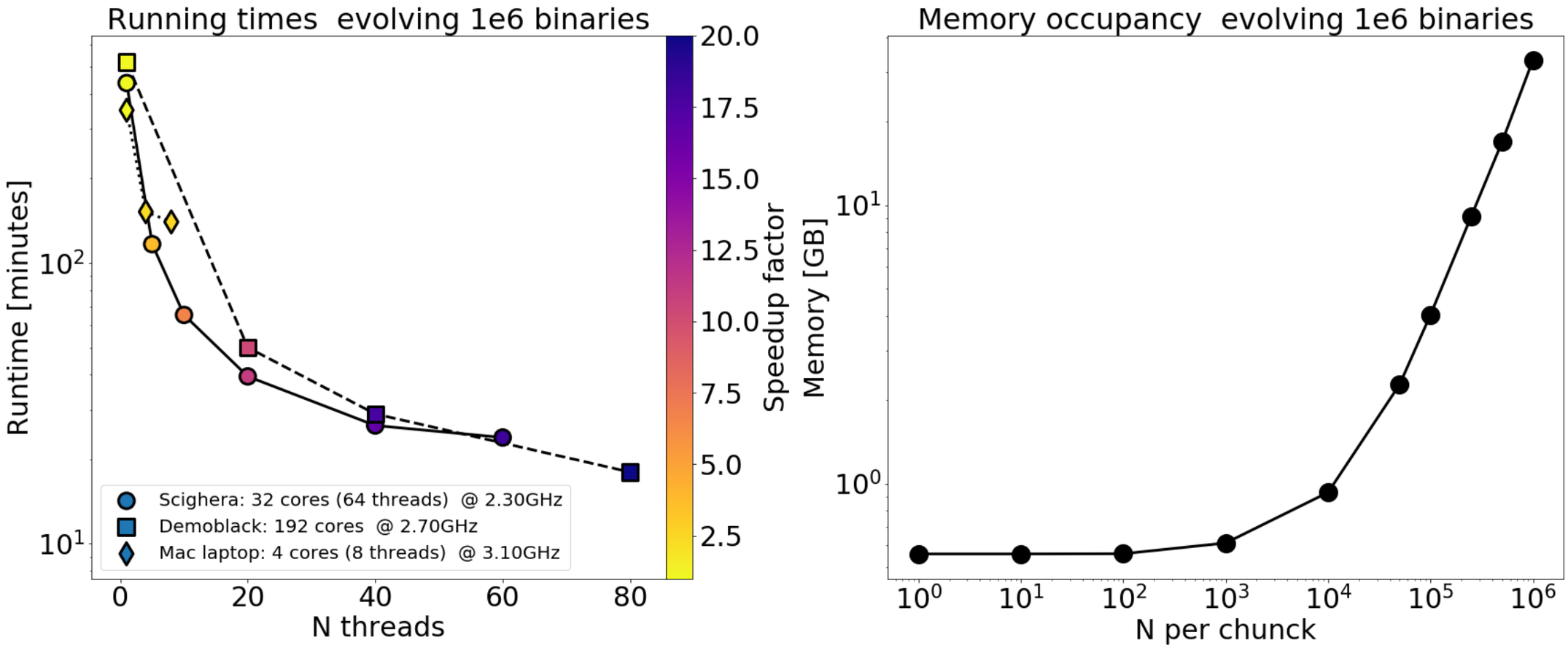
default: 1000

In SEVN2 only Nchunk systems at time are loaded and evolved. After the evolution of these systems is concluded a new set of Nchunk systems is loaded and evolved and so on and so forth until the end of the input file.

Larger value of Nchunk produce slightly faster run at the cost of memory pressure.

Runtime options - run options

The complete list of runtime options can be found here



Notice: **the focus is the curves trend**, the absolute values of Runtime and memory can depend on the detail of the evolution.

- **Use Nchunk as large as possibile, leaving a safe unoccupied memory space** (about 30%). Be careful if you are using a shared machine. Use a number of **Nchunk that is multiple of Nthreads** (to balance the threads work).

- OpenMP seems to not fully exploit the virtual multithreading, therefore the **number of threads should be <= the number of physical cores**.

Runtime options - output options

The complete list of runtime options can be found here

Output folder

-o arg OUTPATH="arg"

arg: path for the folder containing the outputs
default: sevn_ouput

if the folder already exist, its content will be removed.

Output format

-omode arg OMODE="arg"

arg: output format: *ascii, csv, hdf5*

default: csv

hdf5 should be enabled at compile time, see this **slide**

Runtime options - output options

The complete list of runtime options can be found here

Literal phases

-io_literal_phases arg LITPHASES="arg"

arg: bool (true-false)
default: false

If true the phases, remnant types and events will be stored in output as strings, otherwise as integers (see conversion tables for phases, remnant types and events).

Logfile

-io_logfile arg LOGFILE="arg"

arg: bool (true-false)

default: true

If true save the logfile(s) (see …) in output

Runtime options - output options

The complete list of runtime options can be found here

Star properties

-scol arg SCOL="arg"

arg: list of properties separated by semicolons
default: *Worldtime:Mass:Phase:RemnantType*

These are the stellar properties that will be saved in the main output files.

Binary properties

-bcol arg BCOL="arg"

arg: list of properties separated by semicolons
default: *Semimajor:Eccentricity:BEvent*

These are the binary properties that will be saved in the main output files.

Runtime options - output properties

Star properties

The full list of star properties can be found on the wiki

Mass: stellar mass in Msun

MHE: mass of the HE core (includes also the CO core) in Msun

MCO: mass of the CO core in Msun

Luminosity: (bolometric) luminosity of the star in Lsun

Radius: radius of the star in Rsun

Phase: phase of the star (see phases tab, see here)

RemnantType: remnant type (see types tab, see here)

Temperature: temperature of the star in K

Spin: spin of the star (angular velocity over critical angular velocity, see here)

Inertia: moment of the inertia of the star un Msun Rsun^2

Worldtime: elapsed time of the simulation in Myr

Timestep: used timestep in Myr

Runtime options - output properties

Binary properties

The full list of binary properties can be found on the wiki

Semimajor: semimajor axis of the binary in Rsun

Eccentricity: eccentricity of the binare, range [0,1[

Period: orbital period of the binary in days

GWtime: Peters Gravitational Wave decay time scale in Myr (Eq. 5.10 in Peters64)

BEvent: int or str (see here) defining the event happened in a given timestep

BWorldtime: elapsed time of the simulation in Myr (it is equivalent to the star property Worldtime)

BTimestep: used timestep in Myr, (it is equivalent to the star property Timestep)

Runtime options

The complete list of runtime options can be found here

Processes - Winds

-wmode arg WINDSMODE="arg"

arg: '*hurley_wind*' or '*disabled*'
default: '*hurley_wind*'

disabled: disable the process

hurley_wind: Wind formalism from *Hurley+02*

Related params:

alpha -w_alpha double (def 1.5) WALPHA="1.5"

tune the mass accretion rate (Eq. 6 *Hurley+02*)

beta -w_beta double (def 0.125) WBETA="0.125"

tune the wind velocity (Eq. 9 *Hurley+02*)

Runtime options

The complete list of runtime options can be found here

Processes - Supernova kicks

-sn_kicks arg SNKICKS="arg"

arg: '*unified*','*hobbs*','*zeros*'
default: '*unified*'

zeros: disable the Supernova kicks (all kicks will have a module of 0)

hobbs: kicks (module) are taken from a Maxwellian with $\sigma = 265\ \mathrm{km\ s^{-1}}$ (see *Hobbs+05*)

unified: kicks (module) are taken following *Giacobbo+20*. In practice the kick magnitude $V_{\mathrm{kick}} \propto f_{\mathrm{H05}} M_{\mathrm{ej}} M_{\mathrm{rem}}^{-1}$, where f_{H05} is drawn from the hobbs Maxwellian (see above), M_{ej} is the mass of the SN ejecta and M_{rem} is the mass of the remnant.

Runtime options

The complete list of runtime options can be found here

Processes - Supernova (SN) explosion

-snmode arg SUPERNOVA="arg"

arg: '*list*','*rapid*','*delayed*','*compact*'
default: '*list*'

list: use the sn option in the input list, it could be one of the following
rapid: rapid formalism from *Fryer et al. 2012*
delayed: delayed formalism from *Fryer et al. 2012*
compact: compactness formalism from *Mapelli et al. 2020*

Related params (for option "compact"):

Csi25 threshold -sn_compact_csi25_tshold (def -1) SNC25TS="-1"

Compactness ($\xi_{2.5}$) threshold for explosion/implosion (see *Mapelli et al. 2020*). *If -1,* use a stochastic explosion/implosion decision based on the distribution in the bottom panel of Fig. 3 in *Patton&Sukhbold20*

Fallback -sn_compact_fallback (def 0.9) SNCOMPFBS="0.9"

Fallback fraction for explosions

Runtime options

The complete list of runtime options can be found here

Processes - Supernova (SN) explosion - other params

Chandrasekar mass -sn_Mchandra double (def 1.41) MCHANDRA="1.41"

Assumed Chandrasekar mass in Msun, WD with masses larger than this value explodes as SNIa leaving no remnants.

WD/ECSN mass transition H stars -sn_co_lower_ecsn double (def 1.38) SNLOWECSN="1.38"

Mass transition (Msun) for the mass of the CO core to produce a WD or an electron capture SN leaving a NS.

WD/ECSN mass transition He stars -sn_co_lower_ecsn double (def -1) SNLOWECSN="-1"

Same as above, but for pureHe stars. If -1 use the same value of the H-stars

ECSN/ CCSN mass transition H star -sn_co_lower_sn double (def 1.44) SNLOW="1.44"

Mass transition (Msun) for the mass of the CO core to trigger an electron capture SN leaving a NS or to trigger a core collapse SN explosion/implosion leaving a NS, BH or no remnants.

ECSN/ CCSN mass transition He star -sn_co_lower_sn double (def -1) SNLOW="-1"

Same as above, but for pureHe stars. If -1 use the same value of the H-stars

Runtime options

The complete list of runtime options can be found here

Processes - Winds

NOTICE:

The Wind process in SEVN takes into **account only the accretion by winds and the orbital modifications** due to the mass transfer.

The wind mass loss is accounted in the stellar evolution tracks.

It is not possible to introduce a custom wind mass loss!

Runtime options

The complete list of runtime options can be found here

Processes - Common Envelope

-cemode arg CEMODE="arg"

arg: '*energy*' or *'disabled'*
default: '*energy*'

disabled: disable the process

energy: "classical" alpha-lambda energy formalism (see *Hurley+02*)

Related params:

alpha -ce_alpha double (def 5) CEALPHA="5"

Fraction of binding energy used to remove the envelope (Eq. 71 *Hurley+02*)

Lambda -star_lambda double (def -1) CELAM="-1"

Tune the binding energy of the stellar envelope (Eq. 69 *Hurley+02*)
If equal to -1 use the lambda values use the same formalism used in BSE (see the wiki for further options)

Lambda ionization -star_lambda_fth double (def 1) CELAMFTH="1"

Fraction of the ionization energy used in the envelope unbinding in the Clayes+14 formalism (Eq. A6 in Clayes+14)

Notice this parameter is considered only if star_lambda<0

Runtime options

The complete list of runtime options can be found here

Processes - Roche Lobe Overflow (RLO)

-rlmode arg RLMODE="arg"

arg: '*hurley_rl*' or *'disabled'*
default: '*hurley_rl*'

disabled: disable the process

hurley_rl: Roche-lobe overflow formalism by *Hurley+02*

Related params:

Accreted mass fraction -rlo_f_mass_accreted double (def 0.5) RLOMACCR="0.5"

Fraction of the mass loss during the RLO that is accreted on the companion

Eddington factor -rlo_eddington_factor double (def 1) RLOEDD="1"

The accretion on compact objects is limited to this fraction of the Eddington accretion limit

Ang. Momentum loss -rlo_gamma_angmom (def -1) RLOGAM="-1"

Fraction of angular momentum loss from the system during the RLO, special cases:
if -1 the angular momentum is lost from the binary
if -2 the angular momentum is lost from the secondary

Continue

Runtime options

The complete list of runtime options can be found here

Processes - Roche Lobe Overflow (RLO)

Related params:

Circularisation -rlo_circularise bool (def false) RLOCIRC="false"

If true, circularise the binary at the onset of the RLO

Quasi Homogeneous evolution -rlo_QHE bool (def false) SQHE="false"

Enable the quasi-homogeneous evolution after accreting mass during the RLO (other conditions have to be fulfilled, see *Eldrige&Stanway2012*, Sec. 2.2) The stars that are flagged as QHE freeze the evolution of the radius during the Main Sequence, then they become pureHE stars after the Main Sequence.

Continue

Runtime options

The complete list of runtime options can be found here

Processes - Roche Lobe Overflow (RLO)

Related params:

Mass transfer stability (qcrit)

-rlo_stability string or double (def *qcrit_hurley_webbink*) RLOSTABILITY

In the *hurley_rl* formalism the RLO mass transfer is stable if the stars mass ratio (q) is smaller than a given value *qcrit*. The *qcrit* depends on the stellar type of the donor and, for some prescription, of the accretor. The option in SEVN are:

- *number:* if a double is used instead of a string option, this value is used as a constant qcrit.
  - *qcrit_hurley:* qcrit from *Hurley+02*
  - *qcrit_hurley_webbink:* qc from *Hurley+02* with Webbink1988 formalism for giant donors
  - *qcrit_hurley_webbink_shao:* as above + *Shao+21* conditions for BH accretor
  - *qcrit_cosmic_neijssel:* formalism from *Neijssel19* as used in *COSMIC*
  - *qcrit_cosmic_claeys*: *Claeys+14* formalism as used in *COSMIC*

See the wiki (qcrit) for further details

Runtime options

The complete list of runtime options can be found here

Processes - Tides

-tmode arg TIDES="arg"

arg: '*tides_simple*' or '*disabled*'

default: '*tides_simple*'

disabled: disable the process

tides_simple: Equilibrium tides by *Hut1981* (see also *Hurley+02*)

Processes - Gravitational Wave (GW)

-gwmode arg GWMODE="arg"

arg: '*peters*' or '*disabled*'

default: '*peters*'

disabled: disable the process

peters: *GW* orbital decay by *Peters1964*

Related params:

GW activation -gw_tshold double (def 1) GWTSHOLD="1"

Enable the GW process only if GW time scale < gw_tshold*Hubble_time

Runtime options

The complete list of runtime options can be found here

Change of track (see here)

Naked threshold -ev_naked_tshold double (def 1E-4) NAKEDTS="1E-4"

When a star has a mass difference between the *Mass* and *MHE* lower than this parameter (Msun), the stars becomes a pureHE jumping in a new track. If the star is already a pureHE and *MHE* and *MCO* are within this parameter, the star becomes a nakedCO stopping its evolution.

Jump trigger threshold error -jtrack_tshold_dm_rel double (def 0.01) JTDMTSHOLD="0.01"

Minimum relative change in mass to trigger the change of track. E.g. if a star accretes more than 1% of its mass, a change of track is triggered.

Jump max error -ev_naked_tshold double (def 1E-4) JTEMAX="5E-3"

Maximum relative error to reach *convergence* when jumping on a new track. If convergence is not reached the track given the best match (among the tested ones) is chosen. Notice: the best track can be also represented by the current track.

Jump max iterations -jtrack_max_iteration integer (def 0.01) JTMAXITER="10"

Maximum number of iteration to perform searching for the best match. If this limit is reached, the track given the best match (among the tested ones) is chosen. Notice: the best track can be also represented by the current track.

Hints and tips

Running SEVN in background

In order to run a long SEVN run or run SEVN in a server, it is preferable to run SEVN in background

```
nohup ./run.sh <path_to_input_list> >out.log 2>err.log & disown
```

The file *out.log* contains the standard SEVN message in output, while the file *err.log* contains all the Warning and Error messages.

Hints and tips

Monitoring the SEVN run

The standard SEVN output on screen (or on the logfile, see above) reports a counter of evolved and failed stars updated each Nchunk of processed stars:

```
4000020 loaded stars!
Evolving systems:
200000/4000020 (Nfailed:0)
```

A more “thin grained” check can be obtained looking at the number or rows present in the evolved files (or in the error files) inside the SEVN output folder: `wc -l evolved*` `wc -l failed*`

The command(s) will return the number of systems (one per row) evolved (or failed) so far in each thread and their total sum.

```
(base) 🌊 🦊wc -l evolved_*
     571 evolved_0.dat
     561 evolved_1.dat
    1132 total
```

Sevn2 outputs

Output files - overview

```
(base) iogiul@Giulianos-MBP SEVN_run % cd my_output
(base) iogiul@Giulianos-MBP my_output % ls
evolved_0.dat                   logfile_0.dat
failed_initialisation_0.dat     output_0.csv
launch_line.txt                 used_params.svpar
```

A SEVN2 run produce several output files:

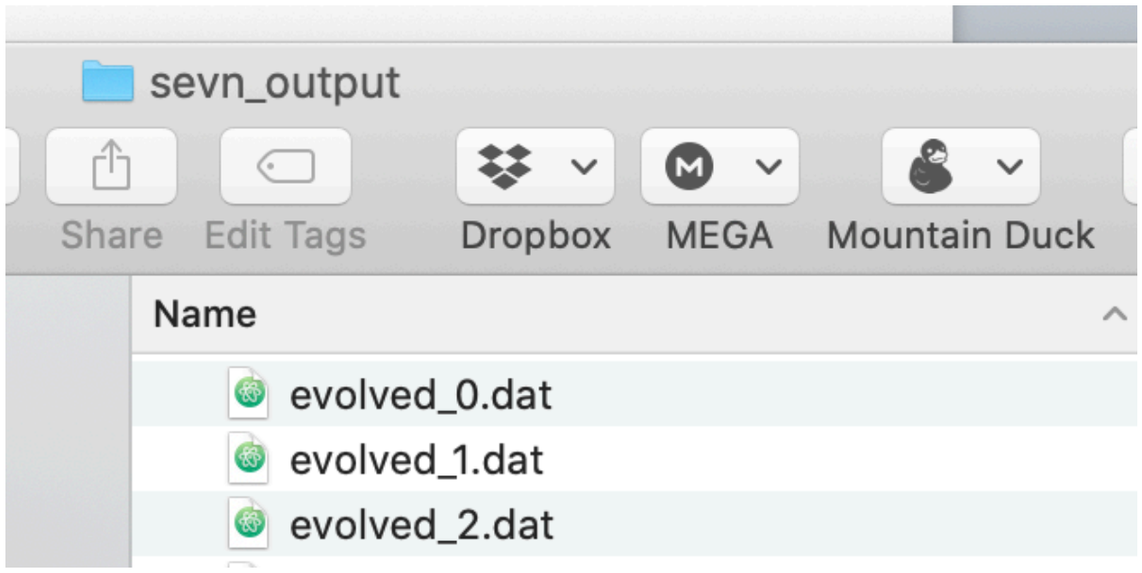
- **Evolved file(s)**: *evolved_xxx.dat*,
  initial properties (input) of the systems that has been successfully evolved
- **Not evolved file(s):** *failed_initialisation_xxx.dat* and *failed_xxx.dat*,
  initial properties (input) of the systems for which the simulation has been halted due to some error.
- **Output file(s)**: *output_xxx.csv* (the extension depends on the runtime option omode), proper output of the simulation
- **Log file(s)**: *logfile_xxx.dat*,
  info on the SSE/BSE, complementary to the output files
- **Summary of the used parameters**: *used_params.svpar*,
  complete list of SEVN parameters used in the simulation
- **Launch line** (only present if the run script has been used): *launch_line.txt*,
  contains the run command actually produced by the run script

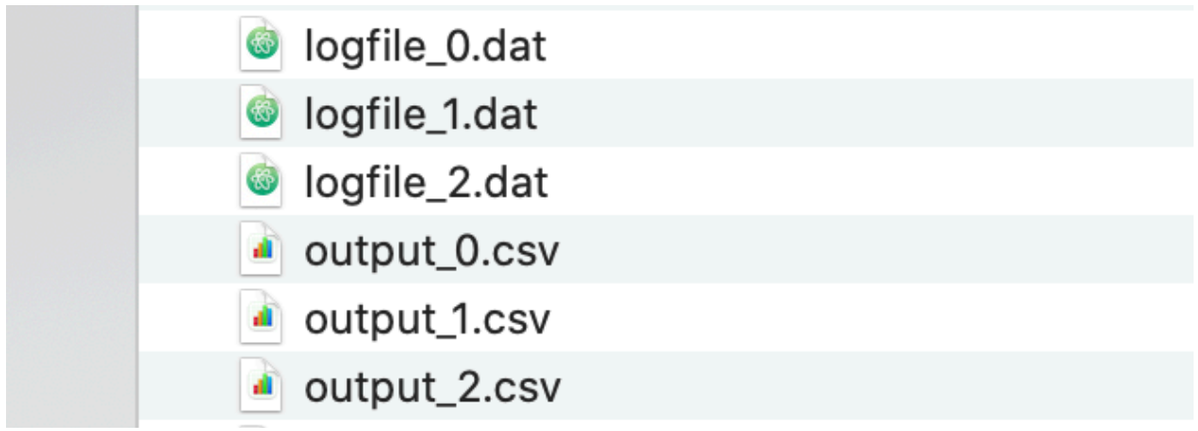
Output files - overview

evolved_xxx.dat, output_xxx.csv, logfile_xxx.dat, failed_xxx.dat

What does xxx mean?

- SEVN exploits cpu-parallelism through multithreading
- Each tread produces its own outputs
- The suffix **xxx** indicates the **thread number**
- The thread number (starting from 0) is printed without leading or trailing zeros





Evolved and failed files:

evolved_xxx.dat, *failed_initialisation_xxx.dat*, *failed_xxx.dat*

Format

- Formatted ascii files
- **Values are separated by white spaces**
- The **first row contains the header.** It starts with the character "#" and contains the column names separate by white spaces.

Content

- Input information of all the evolved (evolved_xxx), failed during evolution (failed_xxx) and failed on initialisation (failed_initialisation_xxx) systems with the addition of unique integers, *ID*, *name* and *Seed.*
- Each row refers to a given system.

evolved_0.dat

#ID	name	Mass_0	Mass_1	Z_0	Z_1
0	438060358735771	30	2.2	0.004	0.004
1	438060358735771	9.013	2.2	0.004	0.004
2	438060358735771	9.014	2.2	0.004	0.004

Evolved and failed files:

evolved_xxx.dat, *failed_initialisation_xxx.dat*, *failed_xxx.dat*

Content - Columns - SSE

1-**ID** [long int]: sequential integer, unique for each binary, it indicates the position of the system in the input file
2-**name** [long int]: random integer, unique for each binary
3-**Mass** [double]: zams mass of the star in Msun
4-**Z** [double]: metallicity
5-**spin** [double]: initial spin of the star
6-**SN** [double]: supernova model for the star
7-**Tstart** [str]: initial age of the star
8-**Tend** [str]: ending time of the simulation
9-**Dtout** [str]: time step for output
10-**Seed** [long int]: seed for random number generation [long int]

Evolved and failed files:

evolved_xxx.dat, *failed_initialisation_xxx.dat*, *failed_xxx.dat*

Content - Columns - BSE

1-**ID** [long int]: sequential integer, unique for each binary, it indicates the position of the system in the input file
2-**name** [long int]: random integer, unique for each binary
3-**Mass_0** [double]: zams mass of the first star in Msun
4-**Z_0** [double]: metallicity the first star
5-**spin_0** [double]: initial spin of the first star
6-**SN_0** [double]: supernova model for the first star
7-**Tstart_0** [str]: initial age of the first star
8-**Mass_1** [double]: same as Mass_0 but for the second star
9-**Z_1** [double]: metallicity the second star
10-**spin_1** [double]: same as spin_0 but for the second star
11-**SN_1** [double]: same as SN_0 but for the second star
12-**Tstart_1** [double]: same as Tstart_0 but for the second star
13-**a** [double]: initial semimajor axis of the binary [double, Rsun]
14-**e** [double]: initial eccentricity of the binary [double]
15-**Tend** [str]: ending time of the simulation
16-**Dtout** [str]: time step for output
17-**Seed** [long int]: seed for random number generation [long int]

Output files: *output_xxx.csv*

Notice: this guide assume that the output format is csv, other options are possibile (see here). The ascii format is similar to the csv except for the column delimiter, it uses white spaces instead of commas.

Format

- Comma Separated Variables ascii files **Values are separated by commas** without extra spaces (e.g. 1,2,hello)
- The **first row contains the header** with columns names separated by commas

output_0.csv

```
1  ID,name,Worldtime_0,Localtime_0,Mass_0,MHE_0,MCO_0,Lumino
2  0,438060358735771,0.000000e+00,7.195606e-02,6.993543e+01,
3  0,438060358735771,3.465333e-01,4.184893e-01,6.959649e+01,
4  0,438060358735771,5.836260e-01,6.555820e-01,6.933650e+01,
```

Output files: *output_xxx.csv*

Content

- Stellar and binary properties (set using the runtime options *scol* and *bcol*, see here) at the chosen times (set using the list or runtime option *dtout*, see here and here).

- Each system is identified by a sequential integer *ID* (original position in the input file) and a unique long integer (*name*). These two values are always the first two columns of the output files.

- The order of the other columns in output depends on the order given in the runtime options *scol* and *bcol*.

- For binary systems the stellar properties (*scol*) are printed before the binary properties (*bcol*). The name of the stellar properties columns have a suffix *_0* or *_1* for the primary and secondary star, respectively.

Output files: *output_xxx.csv*

Content - example

- Assume we set the runtime options:
  - *-scol Mass:Radius:MHE*
  - *-bcol Semimajor:Eccentricity:BWorldtime*
- The columns in output are:

col0-ID: ID of the systems (correspond to the position in the input file)
col1-Name: unique identifier of the system
col2-Mass_0: total mass of the first star
col3-Radius_0: radius of the first star
col4-MHE_0: mass of the He-core of the first star
col5-Mass_1: total mass of the second star
col6-Radius_1: total radius of the second star
col7-MHE_1: mass of the He-core of the second star
col8-Semimajor: semimajor axis of the binary
col9-Eccentricity: eccentricity of the binary
col10-BWolrdtime: time elapsed in the simulations (starting always from 0)

Output files: *output_xxx.csv*

Content - notes

- Outputs can contain *nan* values, they represent undefined values

  - The following columns are **never set to nan**: *Worldtime, Timestep, Phase, RemnantType, BWorldtime, BTimestep, Zams, Event.*

  - **If a star does not exist anymore** (because of a stellar merger or after a SN leaving no remnants), the stellar properties are set to *nan.* In this case *Phase=7* and *RemnantType=-1.* Notice, if at least a star is empty, the binary is considered broken (see below)

  - **If the binary does not exist anymore** (because it is broken or after a stellar merger), the binary parameters are set to *nan.*

Output files: *output_xxx.csv*

Content - notes

- **Timestep:** SEVN2 uses an adaptive timestep schema, the timestep differs from system to system and from step to step considering the same system. In the output, the properties *Timestep* and *BTimestep* indicate the time elapsed in the current timestep.

- **Time properties:** The stellar properties *Worldtime* and *Timestep* are always synched with the binary properties *Bworldtime and BTimestep*. Therefore in each step :
  Worldtime_0=Worldtime_1=BWorldtime and
  Timestep_0=Timestep_1=BTimestep
  Printing in output both of them is redundant

- **Time resolution:** in some cases the timestep is smaller than the output time resolution for the property worldtime. In those cases the columns *BWorldtime* and *Worldtime* could have the same value in two consecutive rows. If this happens, the combination of *Worldtime (BWorldtime)* and *Timestep (BTimestep)* can be used to have a correct time tracking.

Log files: *logfile_xxx.dat*

Format

- “Partially” formatted ascii file.
- Each row contains an **header** part with a fixed number of values (4) + an **info** part with variable number of values.
- The **header** values are separated by “;”
- The **info** values are separated by “:”

Content

- Information about the binary and stellar evolution, complementary to the output files.
- Each row reports a particular event happened in a given star or binary system

```
B;136641911126985;3;MERGER;28.340739;0:8.472e+00:1.530e+00:0.000e+00:3:0:1:6.255e+00:0.000e+00:0.000e+00:1:0:14.727
S;136641911126985;0;SN;30.891598;13.2745:1.92026:1.49299:1.23552:5:507.658:643.773:-181.211:305.113:537.134
```

Log files: *logfile_xxx.dat*

Content - how to read it: Header

HEADER The first part of each row is the header:

Object;name;ID;event;time;

- ***Object:*** **S** indicates a single star, **B** indicates a binary
- ***name:*** object name (long int), notice the stars have the same name of the binary they belong to. It can be used to link the systems to the output and evolved files
- ***ID:*** id of the object (long int). If the object is a star the id is 0 if it is the first star or 1 if it is the second one. It can be used to link the systems to the output and evolved files.
- ***event:*** a label in capital letter indicating the triggering event. All the possible events are listed in the next slide
- ***time:*** time of the simulation (Myr) when the event is triggered.

B;621064598874129;13;CE;3.348641;

A Common envelope (*event*=CE) is triggered in the binary with *name* 621064598874129 and *ID* 13 at *time* 3.348641 Myr.

Log files: *logfile_xxx.dat*

Content - events list

Events list - Star:

- **SN:** Supernova explosion (star)
- **WD:** White Dwarf formation, label
- **HENAKED:** He naked formation
- **CONAKED:** CO naked formation

Events list - Binary:

- **BSN:** Supernova explosion (binary)
- **RLO_BEGIN:** Roche Lobe Overflow begin
- **RLO_END:** Roche Lobe Overflow end
- **CE:** Common Envelope
- **COLLISION:** Collision
- **MERGER**: Merger
- **SWALLOWED:** Swallowed star

Log files: *logfile_xxx.dat*

Content - How to read it: Info

Event - Star: Supernova explosion

Header label: **SN**

Trigger: A star explodes as SN

FORMAT: *d1:d2:d3:d4:i5:d6:d7:d8:d9:d10*

- *d1*: total mass before the SN explosion in Msun
- *d2*: He core mass before the SN explosion in Msun
- *d3*: CO core mass before the SN explosion in Msun
- *d4*: mass of the SN remnant in Msun
- *i5*: remnant type (integer, remnant type)
- *d6*: natal kick, i.e. module of the kick velocity extracted from a Maxwellian in km/s
- *d7*: actual module of the velocity after corrections in km/s
- *d8*: x-component of the corrected kick velocity in km/s
- *d9*: y-component of the corrected kick velocity in km/s
- *d10*: z-component of the corrected kick velocity in km/s

INFO

S;101931734175490;0;SN;11.84;3.53:3.53:2.17:1.23:5:245.96:59.45:0.56:-38.22:45.53

HEADER

d1 d2 d3 d4 i5 d6 d7 d8 d9 d10

Log files: *logfile_xxx.dat*

Content - How to read it: Info

Event - Star: White dwarf formation

Header label: **WD**

Trigger: A star becomes a White dwarf

FORMAT: *d1:d2:d3:d4:i5*

- *d1*: total mass before the SN explosion in Msun
- *d2*: He core mass before the SN explosion in Msun
- *d3*: CO core mass before the SN explosion in Msun
- *d4*: mass of the SN remnant in Msun
- *i5*: remnant type (integer, remnant type)

Example:

S;101931734175490;1;WD;207.419314;0.542482:0.542482:0.483104:0.542482:1

Log files: *logfile_xxx.dat*

Content - How to read it: Info

Event - Star: Neutron star formation

Header label: **NS**

Trigger: A star becomes a Neutron star

FORMAT: *i1:d2:d3:d4:d5*

- i*1*: remnant type (integer, remnant type)
- *d2*: mass of the SN remnant in Msun
- *d3*: Initial magnetic field (Gauss)
- *d4*: Initial angular velocity (1/s)
- d*5*: sin alpha, angle between magnetic axis and rotation axis

Example:

S;101931734175490;1;WD;207.419314;0.542482:0.542482:0.483104:0.542482:1

Log files: *logfile_xxx.dat*

Content - How to read it: Info

Event - Star: Helium naked formation

Header label: **HENAKED**

Trigger: A star loses its Hydrogen envelope remaining with a “naked” helium core. Notice, in this case the interpolating tracks changes. The new tracks comes from new evolution tables based on the evolution of this particular naked Helium objects.

FORMAT: *d1:d2:d3:d4:d5:i6:d7:d8:i9*

- *d1*: total mass before the event in Msun
- *d2*: He core mass before event in Msun
- *d3*: CO core mass before event in Msun
- *d4*: radius of the star before the event in Rsun
- *d5*: radius of the star after the event in Rsun
- *i6*: current stellar phase (integer phase type)
- *d7*: interpolating track Mzams before the event in Msun
- *d8*: interpolating track Mzams after the event in Msun
- *i9*: outcome of the new track matching: 0- track found and convergence reached, 1-track found convergence not reached 2-track not found

Example:

S;32010465718430 1;1;HENAKED;3.21;28.49:25.23:0:221.389:2.4119:4:87.0625:25.2393:0

Log files: *logfile_xxx.dat*

Content - How to read it: Info

Event - Star: CO naked formation

Header label: **CONAKED**

Trigger: A star has lost both its Hydrogen envelope and HE portion of the core remaining with a “naked” Carbon Oxygen core. Notice, in this case the stellar evolution is halted. The star evolves passively until it explodes as SN or becomes a WD.

FORMAT: *d1:d2:d3:d4:d5:d6:d7:i8*

- *d1*: total mass before the event in Msun
- *d2*: He core mass before event in Msun
- *d3*: CO core mass before event in Msun
- *d4*: radius of the star before the event in Rsun
- *d5*: radius of the CO core before the event in Rsun
- d*6*: total mass after the event in Msun
- *d7*: radius of the star after the event in Rsun
- *d8*: current stellar phase (integer phase type)

Example:

S;108335909573341;0;CONAKED;26.57;1.69:1.69:0.90:11.69:0.027:0.90:0.027:6

Log files: *logfile_xxx.dat*

Content - How to read it: Info

Event - Binary: Supernova explosion

Header label: **BSN**
Trigger: One of the star in the (still bound) binary system explodes as SN or becomes a WD

FORMAT: *i1:d2:d3:d4:i5:i6:i7:d8:d9:d10:i11:i12:d13:d14:d15:d16:d17:d18*

- *i1*: id of the first star
- *d2*: mass of the first star before the event in Msun
- *d3*: HE core mass of the first star before the event in Msun
- *d4*: CO core mass of the first star before the event in Msun
- *i5*: phase of the first star before the event (integer phase type)
- *i6*: remnant type of the first star before the event (integer remnant type)
- *i7 - i12*: same as *i1 - i6* but for the second star
- *d13*: semimajor axis length before the event in Rsun
- *d14*: eccentricity before the event
- *d15*: semimajor axis length after the event in Rsun.*
- *d16*: eccentricity after the event in Rsun.*
- *d17*: cosine of the angle between the normal to the old orbital plane (before the event) and the normal to the new orbital plane (after the event).*
- *d18*: module of the velocity of the binary center of mass after the event in km/s.*

* If the binary has been broken by the event, this value is *nan*.

Example:

B;555880334646216;407478;BSN;11.06;0:3.79:3.79:2.38:6:0:1:11.47:0.00:0.00:1:0:228.46:0:160.49:0.42:-0.85:14.72

Log files: *logfile_xxx.dat*

Content - How to read it: Info

Event - Binary: RLO begin

Header label: **RLO_BEGIN**
Trigger: One of the star in the binary system fills the Roche-Lobe and a Roche-Lobe overflow begins

FORMAT:
i1:d2:d3:d4:i5:i6:d7:d8:i9:d10:d11:d12:i13:i14:d15:d16:d15:d16:d17:d18:d19:d20

- *i1*: id of the star filling the RLO (donor star)
- *d2*: mass of the donor star in Msun
- *d3*: HE core mass of the donor star in Msun
- *d4*: CO core mass of the donor star in Msun
- *i5*: phase of the donor star (integer phase type)
- *i6*: remnant type of the donor (integer remnant type)
- *d7*: radius of the donor in Rsun
- *d8*: RL radius of the donor in Rsun
- *i9* - d*16*: same as *i1* - *d8* but for the other star (accretor)
- *d17*: mass ratio q (Mdonor/Maccretor)
- *d18*: critical mass ratio qcrit, if q<qcrit the mass transfer is stable, otherwise unstable
- *d19*: Semimajor axis length in Rsun
- *d20*: Eccentricity

Example:

B;32010465184301;24;RLO_BEGIN;3.21;1:28.58:25.13:0.0:3:0:74.14:71.371:0:53.06:0.0:0.0:7:6:2.17e-04:94.64:0.53:4:218.38:0.004

Log files: *logfile_xxx.dat*

Content - How to read it: Info

Event - Binary: RLO end

Header label: **RLO_END**

Trigger: the RLO started previously comes to and end both because the two stars have merged or because they both stop to fill the RL

FORMAT:
i1:d2:d3:d4:i5:i6:d7:d8:i9:d10:d11:d12:i13:i14:d15:d16:d15:d16:d17:d18:d19:d20

- *i1*: id of the star filling the RLO (donor star)
- *d2*: mass of the donor star in Msun
- *d3*: HE core mass of the donor star in Msun
- *d4*: CO core mass of the donor star in Msun
- *i5*: phase of the donor star (integer phase type)
- *i6*: remnant type of the donor (integer remnant type)
- *d7*: radius of the donor in Rsun
- *d8*: RL radius of the donor in Rsun
- *i9* - d*16*: same as *i1* - *d8* but for the other star (accretor)
- *d17*: total mass lost by the donor during the RLO in Msun
- *d18*: total mass accreted by the acrretor during the RLO in Msun
- *d19*: Semimajor axis length in Rsun
- *d20*: Eccentricity

Example:

B;32010465184301;24;RLO_END;3.21;1:28.49:25.24:0.0:3:0:221.38:71.30:0:53.09:0.0:0.0:7:6:0.0002:94.70:-0.052:0.00015:218.361:0

Log files: *logfile_xxx.dat*

Content - How to read it: Info

Event - Binary: Common Envelope

Header label: **CE**

Trigger: a common envelope evolution is triggered due to a RLO unstable mass transfer or to a stellar collision

FORMAT: *i1:d2:d3:d4:i5:i6:i7:d8:d9:d10:i11:i12:d13:d14:i15*

- *i1*: id of the star that starts the CE (primary)
- *d2*: mass of the primary star in Msun before the event
- *d3*: HE core mass of the primary star in Msun before the event
- *d4*: CO core mass of the primary star in Msun before the event
- *i5*: phase of the primary star before the event (integer, phase type)
- *i6*: remnant type of the primary star before the event (integer, remnant type)
- *i7* - *i12*: same as *i1* - *i6* but for the other star (secondary)
- *d13*: semimajor axis length before the event in Rsun
- *d14*: semimajor axis length after the event in Rsun
- *i15*: 0-if the binary survives after the CE, 1-if the stars coalesce

Example:

```
B;633318987585808;603010;CE;33.25;0:1.43:1.43:0.77:6:0:1:4.44:0.0:0.0:1:0:37.40:7.86:0
```

Log files: *logfile_xxx.dat*

Content - How to read it: Info

Event - Binary: Collision

Header label: **COLLISION**

Trigger: The sum of the radii of the two stars is larger than the stellar separation at the pericentre (R1+R2>a(1-e)) **and** one or both stars are filling the RL

FORMAT: *i1:d2:d3:i4:i5:d6:d7:i8:d9:d10:d11:d12*

- *i1*: id of the star filling the RL (donor star)
- *d2*: mass of the donor star before the event in Msun
- *d3*: radius of the donor star before the event in Rsun
- *i4*: phase of the donor star (integer phase type)
- *i5-i8*: same as *i1* - *i4* but for the other star
- *d9:* Semimajor axis length before the in Rsun
- *d10*: Eccentricity before the event
- *d11:* RL radius in Rsun of the donor star before the event (ID=0)
- *d12:* RL radius in Rsun of the accretor star before the event (ID=0)

Example:

B;19427287500386 8;26;COLLISION;6.09;0:29.71:41.07:4:1:0.5:2.12e-06:7:57.47:0.66:39.98:6.91

Log files: *logfile_xxx.dat*

Content - How to read it: Info

Event - Binary: Merger

Header label: **MERGER**

Trigger: Two stars merge after a collision or a unstable RLO.

FORMAT: *i1:d2:d3:d4:i5:i6:i7:d8:d9:d10:i11:i12:d13*

- *i1*: id of the star that survives after the merger (accretor)
- *d2*: mass of the accretor star in Msun before the event
- *d3*: HE core mass of the accretor star in Msun before the event
- *d4*: CO core mass of the accretor star in Msun before the event
- *i5*: phase of accretor star before the event (integer phase type)
- *i6*: remnant type of accretor star before the event (integer remnant type)
- *i7* - *i12*: same as *i1* - *i6* but for the other star (donor)
- *d13:* final mass of the accretor star after the merger

Example:

```
B;495263899355111;1;MERGER;31.52;0:8.21:1.40:0.0:3:0:1:4.32:0.0:0.0:1:0:12.5368
```

Log files: *logfile_xxx.dat*

Content - How to read it: Info

Event - Binary: Swallowed

Header label: **SWALLOWED**
Trigger: during an unstable RLO the star filling the RL is completely destroyed, e.g. the mass lost is the total stellar mass and neither a merger nor a CE is triggered.

FORMAT: *i1:d2:d3:d4:i5:i6:i7:d8:d9:d10:i11:i12*

- *i1*: id of the destroyed star
- *d2*: mass of the destroyed star in Msun before the event
- *d3*: HE core mass of the destroyed star in Msun before the event
- *d4*: CO core mass of the destroyed star in Msun before the event
- *i5*: phase of the destroyed star before the event (integer phase type)
- *i6*: remnant type of the destroyed star before the event (integer remnant type)
- *i7* - *i12*: same as *i1* - *i6* but for the other star

Example:

```
B;208699110706408;652062;SWALLOWED;23.88;1:1.38:1.380:1.38:6:0:0:1.23:0.00:0.0:7:5
```

Summary of used parameters: *used_params.svpar*

Format

- Formatted ascii file
- The first row contains the header “#USED PARAMS”
- The other rows are formatted as:
  parameter_name : value //short_description

Content

- This file stores the values of all the SEVN parameters used in the simulation.

used_params.svpar

```
#USED PARAMS
ce_alpha:                       5                                       //[N][PS] alpha in binding energy (
ce_kce:                         1                                       //[N][PS] Fraction of non core mass
ce_knce:                        1                                       //[N][PS] Fraction of non core mass
ev_Nchunk:                      1000                                    //[N][PS] Evolve Nchunk at time
```

Launch line: *launch_line.txt*

This file is included in the outputs only if SEVN has been run using the run script.

Format

- Text file containing two parts separated by a semicolon.

Content

- The first part contains info about the date and time of the SEVN execution and the name of users that run it.
- The second part contains the runs command actually executed through the run script

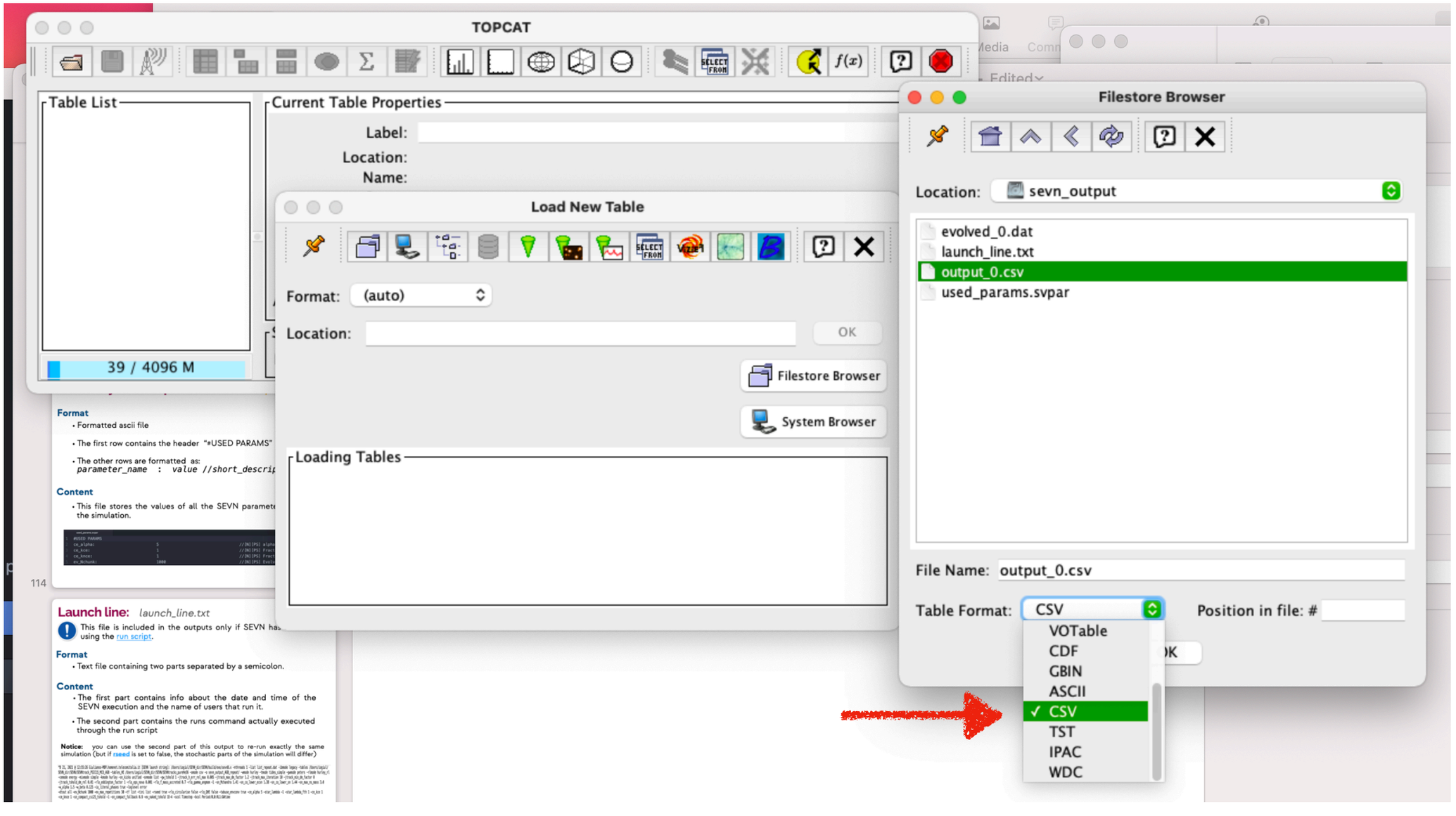
Notice: you can use the second part of this output to re-run exactly the same simulation (but if **rseed** is set to false, the stochastic parts of the simulation will differ)

```
^B 21, 2021 @ 12:55:26 Giulianos-MBP.homenet.telecomitalia.it [SEVN launch string]: /Users/iogiul/SEVN_dir/SEVN/build/exe/sevnB.x -nthreads 1 -list list_repeat.dat -ibmode legacy -tables /Users/iogiul/
SEVN_dir/SEVN/SEVNtrack_PSIC15_MCO_AGB -tables_HE /Users/iogiul/SEVN_dir/SEVN/SEVNtracks_pureHe36 -omode csv -o sevn_output_AGB_repeat/ -wmode hurley -tmode tides_simple -gwmode peters -rlmode hurley_rl
-cemode energy -mixmode simple -kmode hurley -sn_kicks unified -snmode list -gw_tshold 1 -jtrack_h_err_rel_max 0.005 -jtrack_max_dm_factor 1.2 -jtrack_max_iteration 10 -jtrack_min_dm_factor 0
-jtrack_tshold_dm_rel 0.01 -rlo_eddington_factor 1 -rlo_eps_nova 0.001 -rlo_f_mass_accreted 0.7 -rlo_gamma_angmom -1 -sn_Mchandra 1.41 -sn_co_lower_ecsn 1.38 -sn_co_lower_sn 1.44 -sn_max_ns_mass 3.0
-w_alpha 1.5 -w_beta 0.125 -io_literal_phases true -loglevel error
-dtout all -ev_Nchunk 1000 -ev_max_repetitions 30 -tf list -tini list -rseed true -rlo_circularise false -rlo_QHE false -tabuse_envconv true -ce_alpha 5 -star_lambda -1 -star_lambda_fth 1 -ce_kce 1
-ce_knce 1 -sn_compact_csi25_tshold -1 -sn_compact_fallback 0.9 -ev_naked_tshold 1E-4 -scol Timestep -bcol Period:RL0:RL1:GWtime
```

Hints and tips

Reading and handling the output files

- The software TOPCAT can be used to give a first quick look to the outputs. It reads both ascii and csv files, just use the right extension when loading the file



Hints and tips

Reading the output files

- The Python package PANDAS can read output files both on ascii,/csv and hdf5 format.

CSV

```
import pandas as pd
df = pd.read_csv(“path_to_output_file”)
```

See example 7

ASCII

```
import pandas as pd
df = pd.read_csv(“path_to_output_file”,  sep=r"\s+") //read file
df=df.rename(columns={'#ID':ID})   //rename ID column to remove
the #
```

HDF5

```
import pandas as pd
df = pd.read_hdf(“path_to_output_file”) //read file
```

Pandas read functions return a pandas **dataframe**

Hints and tips

Filtering the output files

- The Python package PANDAS can read output files both on ascii,/csv and hdf5 format.
- PANDAS dataframes can be filtered creating boolean list through boolean operations

E.g. filter bound binary black holes See example 7

```
#Dataframe can be filtered using boolean operations on their columns as done with nunpy array,
#For example, let assume we want to retrieve all the BH-BH bound #binaries.
#Remember that the BHs hav RemnantType=6 and bounded binaries have #not nan Semimajor

idxBHBH=(df.RemnantType_0==6) & (df.RemnantType_1==6) & (df.Semimajor.notnull())

#now use the above index to extract a new dataframe of bounde BHBH binaries

dfbhbh=df[idxBHBH]
```

Hints and tips

Reading the evolved files

- PANDAS can also read the evolved (and failed) files

```
import pandas as pd
df = pd.read_csv("path_to_evolve_file", delimiter=r"\s+")
df= df.rename(columns={'#ID':ID})   //rename ID column to remove
the #
```

- It could be useful to rename the columns name to avoid to have same names of the output files (this is useful when a join of the two files are needed, see next slide)

```
df = df.rename(columns={"Mass_0":"Mzams_0", "Mass_1":"Mzams_1",
"a":"Semimajor_ini","e":"Eccentricity_ini"})
```

Hints and tips

Join evolved and output files

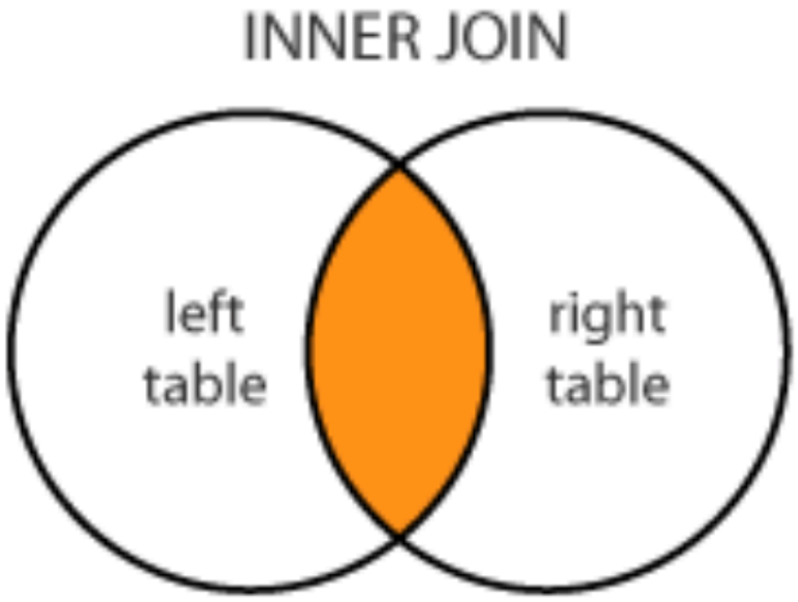
- Sometimes a combination of data in the output and in the evolved files is needed. For example, we want to add information of the initial Mzams and initial orbital properties for all BHBH binaries. The PANDAS method *merge* handles all the possible joins among two data frame.
- Assume we want to merge two dataframes (df_left, df_right) joining together the information of data sharing the same value on a given column (e.g. an ID), this is an inner join.

```
df_merged = df_left.merge( df_right, on="ID",  how="inner", suffixes=("_left","_right") )
```

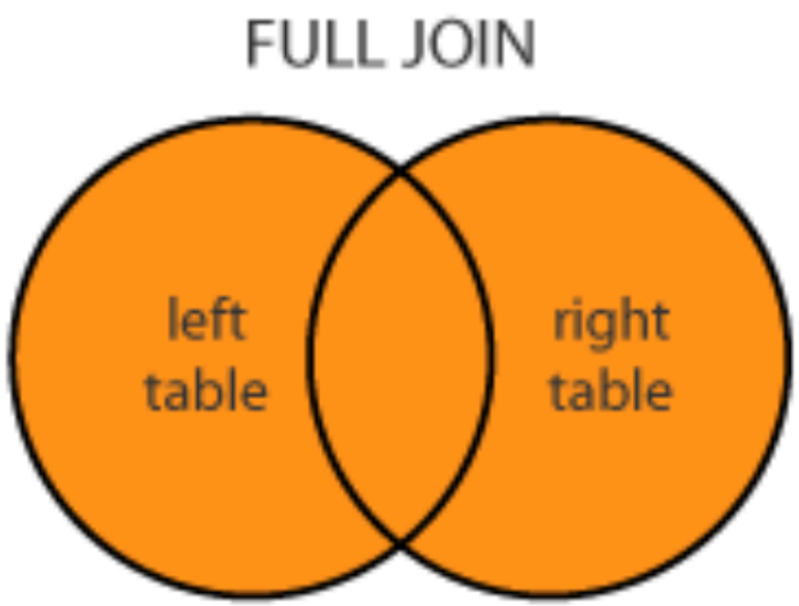
- **on**: column(s, can be a list of columns) to match during the merge of the two tables. The colum(s) has(have) to be present in both the tables
- **how:** type of join to use, see documentation here and the next slide
- **suffixes:** columns with the same name in the two tables (not used in on) will be renamed adding these suffixes.

Hints and tips

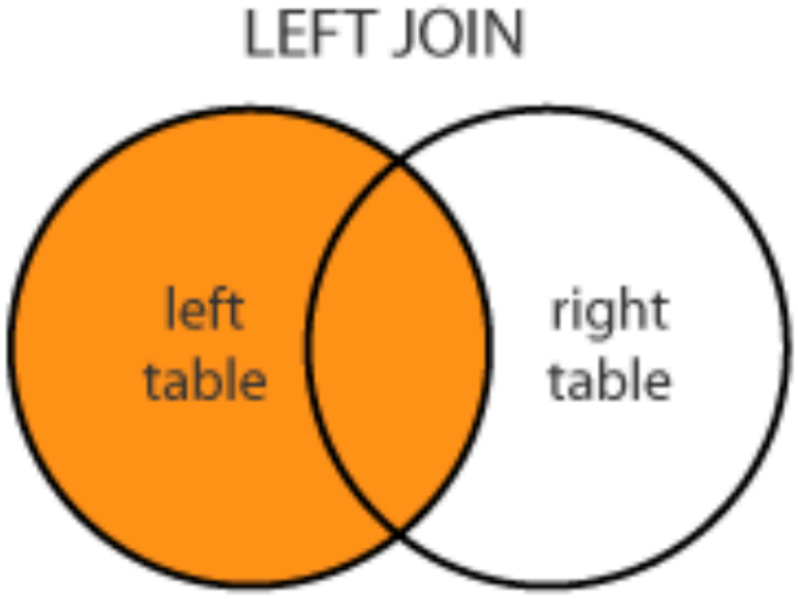
Join evolved and output files - join types



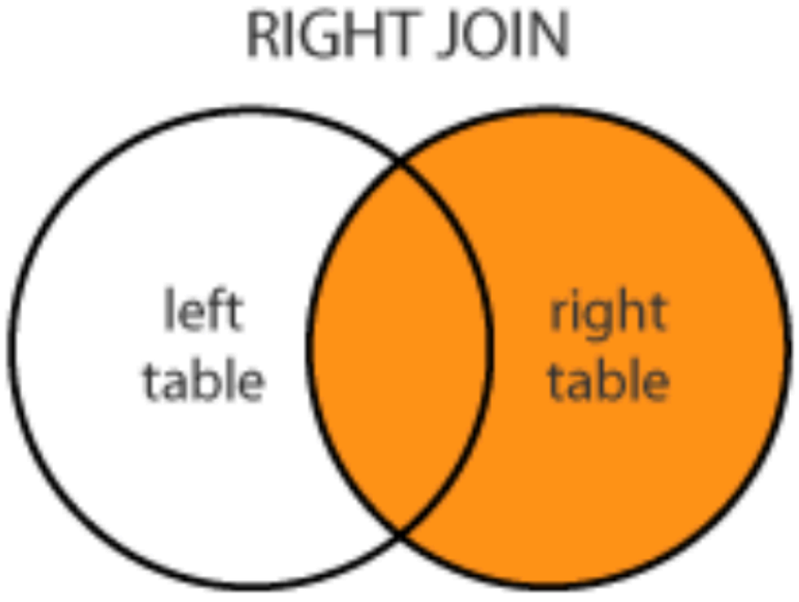
how: “inner”



how: “outer”



how: “left”



how: “right”

Hints and tips

Join evolved and output files - example

- Assume we have a dataframe containing all the binary black holes filtered from the output of a SEVN run (dfbhbh, see here). We want to add to the dataframe the info stored in the evolved output file.

Load the evolved file

```
evolved_path="../sevn_outputs_example/sevn_output_single/evolved_0.dat"
dfe = pd.read_csv(evolved_path,  sep=r"\s+") #df is a pandas dataframe
dfe=dfe.rename(columns={"#ID":"ID"})  #rename ID column to remove the #
#It could be useful to rename the columns name to avoid to have same names
of the output files
dfe = dfe.rename(columns={"Mass_0":"Mzams_0",
"Mass_1":"Mzams_1","a":"Semimajor_ini","e":"Eccentricity_ini"})
```

Join the bhbh dataframe with the evolved file

```
merged_df =dfbhbh.merge(dfe,on=["ID","name"],how="inner",suffixes=("","_"))
```

Notice, in principle just one column among *"ID"* and *"name"* is needed to match the two tables. We used both the column to avoid to have duplicates column in the final merged catalogue (remember the columns used in on, will not be replicated)

See example 8

Hints and tips

Reading and handling multiple files

- When SEVN is used with multithreading parallelisation, the outputs are split in N files where N is the number of used threads.

- Outputs can be gathered loading each single file with PANDAS and then concatenating the daframes with the method concat

```
import pandas as pd
df_0 = pd.read_csv(“output_0.csv”)
df_1 = pd.read_csv(“output_1.csv”)
df_2 = pd.read_csv(“output_2.csv”)
final_df = pd.concat([df_0,df_1,df_2])
```

- Inside the SEVN folder pyscript there is the script *pymake_unique_file*, it can be used to automatically merge the splitted output adding also information from the evolved and log files, its basic usage is:

```
./pymake_unique_file <path_to_the_output_folder> (e.g. sevn_ouput)
```

There are other options that can be set directly inside the script. Give a look

See example 9

Hints and tips

Reading and handling multiple files: DASK

- An other option is to use the python package DASK. The purpose of DASK is to provide parallelism to python data structures. In particular the DASK DataFrame mimics the Pandas data structure and methods.
- Most of the pandas methods can be called in dask using the same formalism. Moreover, some of them have “augmented functionalities”, the read_csv method can automatically handle split files.

```
import dask.dataframe as dd
dt = dd.read_csv(“output_*.csv”) #it will automatically load
#output_0.csv, output_1.csv etc. etc.
```

- DASK is based on the lazy execution concept, in practice the operations on dask dataframes (including the reading of files) are stored (in graphs), but not evaluated until the method compute() is explicitly called. The compute() call on a dask dataframe returns a pandas dataframe.

```
df = dt.compute() #df is a pandas dataframe containing all the
#data of the split output_file in a unique dataframe
```

Hints and tips

Reading and handling multiple files: DASK

- In order to exploit the dask dataframe, the general suggestion is to perform all the operation like filtering, joining using dask and only at end call `compute()` to obtain the pandas `dataframe`.

For example: load the output files and filter the binary black holes

```
import dask.dataframe as dd

dt = dd.read_csv("senv_output/output_*")

idxBHBH=(dd.RemnantType_0==6)
         & (dd.RemnantType_1==6)
         & (dd.Semimajor.notnull())

dtbhbh=dt[idxBHBH]

df = dtbhbh.compute()
```

See example 10

Hints and tips

Reading and handling logfiles: RegEx

Reg[ular]
Ex[pression]*

- Logfiles are “partially” formatted (e.g. number of “columns” changes from event to event) therefore modules as numpy or pandas are not the beste choice to handle them.
- A good alternative to manually loading and analysing the information row by row is to use *RegEx* and its python implementation re (built-in module, not need to install it)
- *“A Regular Expression is a sequence of characters that specifies a search pattern. Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation”* (from Wikipedia). A tutorial for writing RegEx can be found here: http://regular-expressions.info.
- The python module re uses regular expressions to search for pattern in string and retrieve information from them.

Hints and tips

Reg[ular] Ex[pression] *

Reading and handling logfiles: RegEx

Example: RegEx and re to get the all SN kick velocities (module) from the logfile.

The information about the kick velocity is inside the SN event in the logfile:

INFO

S;101931734175490;0;SN;11.84;3.53:3.53:2.17:1.23:5:245.96:59.45:0.56:-38.22:45.53

HEADER

d1 d2 d3 d4 15 d6 d7 d8 d9 d10

In particular, the info we want to get is in *d7* (59.45 km/s in the example above)

1. **First, create the RexEx string**, we can exploit the fact that the event identification is unique

```
regex_str=";SN;[+-]?\d+?[.]?\d+?;(?:[+-]?\d+?[.]?\d*?:){6}([+-]?\d+?[.]?\d*?):"
```

RegEx strings can look very cryptic at the first glance, essentially they include a list of proper characters and meta-characters with special meaning.

The site https://regex101.com is a very useful tool to both learn and test on the fly RegEx strings. It contains also a section where all the parts of the expression are explained.

Hints and tips

Reg[ular] * Ex[pression]

Reading and handling logfiles: RegEx

Example: RegEx and re to get the all SN kick velocities (module) from the logfile.

The information about the kick velocity is inside the SN event in the logfile:

INFO

S;101931734175490;0;SN;11.84;3.53:3.53:2.17:1.23:5:245.96:59.45:0.56:-38.22:45.53

HEADER

d1 d2 d3 d4 15 d6 d7 d8 d9 d10

In particular, the info we want to get is in *d7* (59.45 km/s in the example above)

1. **First, create the RexEx string**, we can exploit the fact that the event identification is unique

```
regex_str="SN;.*?;(?:.*?:){6}(.*?):"
```

;SN; matches the ;SN; part in the string, this identify a row with a SN event

.*? matches the second part of the header (the time), this string is essentially getting any character up to the ;. . matches any character, *? matches the previous token between 0 and unlimited times, as few times as possible.

(?:.*?:){6} matches the first 6 entries of the info part (d1-d6),
(?: …) identify a non-capturing group, i.e. considered what is inside as a group but not take in in output, {6} means replicate six times the previous group

(.*?): matches the 7th entry (d7),
(…) identify a capturing group, i.e. considered what is inside as a group, what it is matched inside the parenthesis is captured for the output

Hints and tips

Reg[ular] Ex[pression] *

Reading and handling logfiles: RegEx

Example: RegEx and re to get the all SN kick velocities (module) from the logfile.

The information about the kick velocity is inside the SN event in the logfile:

S;101931734175490;0;SN;11.84;3.53:3.53:2.17:1.23:5:245.96:59.45:0.56:-38.22:45.53

HEADER | INFO

d1 d2 d3 d4 15 d6 d7 d8 d9 d10

In particular, the info we want to get is in *d7* (59.45 km/s in the example above)

2. **Import re**

```
import re
```

3. **Apply the *findall* method,** to the entire logfile (it has to be read)

```
logfile_path="sevn_output/logfile_0.dat"
with  open(logfile_path,"r") as fo:
    ma=re.findall(regex_str,fo.read())
#ma is a list of all the matched output (they are all strings)
```

4. **Store it in a numpy array**

```
import numpy as np
vkicks=np.array(ma,dtype=float)
```

See example 11

Hints and tips

Reg[ular] Ex[pression]*

Reading and handling logfiles: RegEx

Example: RegEx and re to count the number of CE for each binary of compact objects and add the information to the final output

1. **First, create the RexEx string**, we are interested only on counting CE events, therefore we need just to match the label *CE* in the header of each row and get the ID of the binary

```
regex_str=r'B;\d+;(\d+);CE;'
```

2. **Match and get the ID**,

```
logfile_path=“sevn_output/logfile_0.dat”
with  open(logfile_path,"r") as fo:
    ma=re.findall(regex_str,fo.read())
#ma is a list of all the matched output (they are all strings)
```

3. **Create a Dataframe and group by ID to count the number of CE events**

```
#Create a dataframe with all the ID obtained in the re findall
BID = pd.DataFrame({'ID':np.asarray(ma,dtype=int)})


#Group by ID and count occurences producing a dataframe with the
#column ID and the column NCE containing the number of CEs
BID= BID.groupby(['ID']).size().reset_index(name='NCE')
```

Hints and tips

Reg[ular] * Ex[pression]

Reading and handling logfiles: RegEx

Example: RegEx and re to count the number of CE for each binary of compact objects and add the information to the final output

4. **Load the output file and filter it**, we are interested only on systems ending with a a bounded binary of compact objects (BHBH, NSNS, BHNS)

```
#Load the output file
output_path="../sevn_outputs_example/sevn_output_singleoutput_0.csv"
df = pd.read_csv(output_path)

#Filter it
idx_compact_primary= (df.RemnantType_0==4) | (df.RemnantType_0==5) |
(df.RemnantType_0==6)

idx_compact_secondary= (df.RemnantType_1==4) | (df.RemnantType_1==5) |
(df.RemnantType_1==6)

idxcomp=(idx_compact_primary) & (idx_compact_secondary) &
(df.Semimajor.notnull())

dfcomp=df[idxcomp]
```

Hints and tips

Reg[ular] Ex[pression] *

Reading and handling logfiles: RegEx

Example: RegEx and re to count the number of CE for each binary of compact objects and add the information to the final output

5. **Merge the two dataframe**

```
dfmerged=pd.merge(dfcomp,BID,on='ID',how="left")
```

Notice, we use how="left" because because we want that all the binary in the dfcomp dataframe are included in the final dataframe (see here). The one with 0 CE events will have a NaN in the NCE column.
We will set this NaN to 0 in the following rows

```
dfmerged['NCE'] = dfmerged['NCE'].fillna(0)

dfmerged = dfmerged.astype({"NCE": int})
```

See example 12

Hints and tips

Reading and handling logfiles: RegEx

Reg[ular]
Ex[pression]*

Other examples of regex usage can be found in the wiki page:
Hints and tips