A Short(ish) Guide to Making a Galaxy

Northwestern CIERA — version 1.0

Last updated July 9, 2020

Contents

1 Overview **2**
1.1 What you can find in this guide . . . 2
1.2 What exactly is COSMIC? . . . 3

2 Using COSMIC **5**
2.1 Installation . . . 5
2.2 Generating a fixed population [?] . . . 5
2.3 Evolving binaries yourself [?] . . . 6
2.4 Scaling the fixed population to an astrophysical one [?] . . . 8
2.4.1 Gravitational Wave Evolution . . . 11

3 Understanding COSMIC **13**
3.1 Understanding the theory . . . 13
3.1.1 Metallicity . . . 13
3.1.2 Star Formation History (SFH) . . . 14
3.1.3 Binary Stellar Evolution (BSE) . . . 15
3.1.4 Initial Mass Function (IMF) . . . 15
3.1.5 Signal-to-Noise Ratio (SNR) . . . 15
3.2 Understanding the inputs . . . 16
3.2.1 The `Params.ini` file . . . 16
3.2.2 What does `kstar` refer to? . . . 20
3.3 Understanding the outputs . . . 21
3.3.1 The `bpp` dataframe . . . 22
3.3.2 The `conv` dataframe . . . 22
3.3.3 The `bcm` dataframe . . . 23

A Running COSMIC on Quest **24**
A.1 Initializing COSMIC for the first time . . . 24
A.2 Running COSMIC . . . 24
A.3 Miscellaneous computer things . . . 26

B Documenting my attempts at a fiducial LISA galaxy **28**
B.1 Objectives . . . 28
B.2 Detailed process . . . 28

So you'd like to create your very own population of binary stars, but COSMIC seems a bit confusing? No worries, I'm here to help. I haven't helped develop COSMIC, but I stumbled around the documentation and the source code long enough to be able to help you out a bit along the way!

As I started exploring what COSMIC [1] had to offer, I put together a few tutorials that are available at the following address:

`https://drive.google.com/drive/folders/1VwAFBOZOtb-vC316N76sPbxdqIilWcj4?usp=sharing`

Hence, I have marked with an "[?]" the sections in this document can be accompanied by those tutorials for a little extra support.

1 Overview

1.1 What you can find in this guide

In section 1 – this one – you'll find a quick description of what COSMIC's all about, broad definitions of some of the main aspects of binary population synthesis, and whatever this is.

Section 2 has subsections corresponding to a handful of simple enough things you can do with COSMIC, like how to generate a standard fixed population for example. Some of them are accompanied by a Google Colab Tutorial [?]. Please note that those are mostly here to help you grasp what COSMIC can do broadly. Depending on your goals, you should adapt it to the kind of synthesized population you're looking for and the official COSMIC documentation [1] should be a great guide for this.

Section 3 goes in depth about some COSMIC-specific lingo, what are the inputs & outputs, as well as how to generally customize everything to your specific needs.

You then have the appendices:

Appendix A goes over the specifics of using COSMIC on Northwestern's high performance computing facilities, Quest.

Finally, appendix B is especially interesting (at least to me) : it details the process to getting a population of Milky Way LISA sources. It involves discussions of how COSMIC can be tailored to output the most accurate possible datasets for your purposes.

Overall, the best use I can hope for this guide is to help convey the basic vocabulary pertaining to COSMIC's functioning so you can feel like you have the tools to ask more specific questions about how to tailor it best to your interests.

1.2 What exactly is COSMIC?

COSMIC is short for **C**ompact **O**bject **S**ynthesis and **M**onte Carlo **I**nvestigation **C**ode. You can view it as a fast and efficient toolbox designed for binary population synthesis. It is especially created for the purpose of generating realistic Milky Way binary population. It can evolve binary populations using a Binary Stellar Evolution (BSE) code, directly adapted from Hurley et al's 2002 Fortran BSE code [3] to include much-needed updates to both the code structure and physics behind it.

Some new physics: The legacy BSE code around which COSMIC is built is a bit outdated – and not only because it's in Fortran. A number of modifications were made to account for some new physics in binary evolution that is especially important in compact binary formation. A detailed discussion of the astrophysics "updates" to the code is included in Breivik+2019 [2]. Additionally, you are given the option to use a whole array of different BSE models currently out in the literature that can help establish a stronger understanding of the differences between them.

A flexible and friendly interface: A great feature of COSMIC is how flexible it is. Its user-friendly Python interface can be used to evolve binaries with a single line of code. It's made to account for different possible models in binary stellar evolution. How adaptable it is makes it a bit hard to present a comprehensive, yet digestible user manual, but I'll be trying my best throughout.

A faster and smarter approach to large population synthesis: The real advantage that COSMIC has over most other binary population synthesis codes is its statistical approach to synthesis itself. Instead of generating a full population of initial binaries outright to evolve them one by one, which is time-consuming and relatively inefficient when you get to realistic numbers for a galactic population, COSMIC takes the smarter route of relying on Monte Carlo statistical estimates to extrapolate what the **fixed population** should look like. The process is explained briefly in Fig. 1.1. This also allows to fully grasp the behavior of binary parameters all the way into the tails of the distribution.

What is the fixed population? You can find below the schematic for the process COSMIC uses to generate a fixed population, COSMIC's main output.

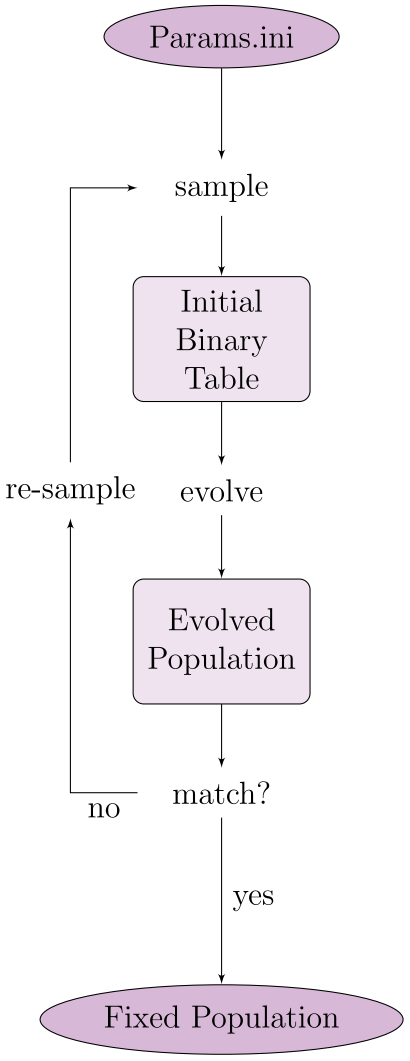


Figure 1.1 Understanding COSMIC's process to getting you that fixed population.

It is a statistical representative of what you'd expect to get out of given star formation parameters (like Star Formation History **SFH** (3.1.2) or Initial Mass Functions **IMF**) and binary stellar evolution process (**BSE**, see section 3.1.3). Once that fixed population has been generated (which takes anywhere from a couple of hours up to a couple of days in general), you can sample it quite fast.

What that means is that you can then use some of the modules incorporated into COSMIC to transform that statistical population into an astrophysical one, by scaling it to a desired number of binaries or total mass and by assigning each binary a galactic location and orientation consistent with our Milky Way.

2 Using COSMIC

This section focuses on specific things you can do with COSMIC. You may not have any need for most of them, but having them laid out here might be helpful in your journey to understanding what COSMIC is all about.

2.1 Installation

COSMIC is constantly being worked on and improved (at least, while I'm writing this) – you might therefore decide to either work with the latest public release of the code (the current official version), or the un-official version, with the very latest beta features. For info on how to get that one, see A.3.

There are some additional subtleties when using COSMIC on Quest, Northwestern's very own high-performance computing cluster. If that's a resource you plan on using, I'm also including a Quest-specific user guide as appendix A.

2.2 Generating a fixed population [?]

You get the fixed population by using the executable `cosmic-pop` and specifying a couple of details. Namely, the parameters that `cosmic-pop` needs to run are as follows:

```
cosmic-pop --final-kstar1 10 14 --final-kstar2 10 14 --inifile
   Params.ini --Nstep 5000 --Niter 10000000 -n 1
```

Listing 2.1 Sample command line for the fixed population

1. `--inifile`

   You just need to have that `Params.ini` file in the directory you're currently working in. It indicates to `cosmic-pop` what models of BSE[1], metallicity, or possibly SFH[2] you want to use. You'll find all details about what's in the file in section 3.2.1.

2. `--final_kstar1` and `--final_kstar2`

   This option lets you decide what population of stars you want out of this. Are you interested in Main Sequence stars? Helium White dwarfs? Black holes? You decide.

   The executable line will only accept single numbers (e.g.: `--kstar1 1`) or ranges (e.g.: `--kstar1 1 5`). See section 3.2.2 for details as to what different values for `kstar` mean.

3. `--Nstep`, `--Niter` and `-n`

   Now, that last part tells COSMIC to use 1 processor (`-n`) to evolve a maximum of 10^7 systems (`--Niter`) and check in every 5000 systems (`--Nstep`) to track how the shape of the distributions of the parameters specified in convergence-params change.

[1]Binary Stellar Evolution: what do you want your recipe to be to evolve your baby stars to grown-up stars? See section 3.1.3 for more info.

[2]Star Formation History: when was your population was actively forming stars? See section 3.1.2.

2.3 Evolving binaries yourself [?]

Though the main tool that COSMIC provides is the executable `cosmic-pop` and its output the fixed population (as outlined in Fig. 1.1), COSMIC comes with a multitude of modules and sub-tools that you can call independently of the whole `cosmic-pop` process.

COSMIC has at its (and your) disposition a whole class of functions to evolve your own set of binaries.

Evolving a single binary You can evolve a single binary of your choice, as long as you provide your chosen binary stellar evolution (BSE) models as well. You'll need to (1) specify some initial parameters and park them in an `InitialBinaryTable` and (2) translate your favorite BSE models into a `BSEdict`, which are both in a language that COSMIC understands.

```
from cosmic.sample.initialbinarytable import InitialBinaryTable
single_binary = InitialBinaryTable.InitialBinaries(m1=85.543645, m2
    =84.99784, porb=446.795757, ecc=0.448872, tphysf=13700.0, kstar1=1,
    kstar2=1, metallicity=0.002)
```

Listing 2.2 How to translate initial conditions into a format understood by COSMIC

The required initial parameters are understood as follows:

1. `m1` and `m2`, or the ZAMS masses of the primary and secondary stars in [$M_\odot$]
2. `porb`, the initial orbital period in days
3. `ecc`, the initial eccentricity
4. `tphysf`, or the total evolution time in Myr
5. `kstar1` and `kstar2`, the initial stellar types of each star – see table 3.2.
6. `metallicity` fraction, knowing that e.g. $Z_\odot = 0.02$

The online tutorial on *Evolving your own binary* [?] features an already-made BSE dictionary that you can copy as is.

Run the following line to make your binary undergo binary stellar evolution as you defined it in `BSEdict`. If you can't remember what `bpp`, `bcm` or `initC` are, see section 3.3.

```
from cosmic.evolve import Evolve
bpp, bcm, initC = Evolve.evolve(initialbinarytable=single_binary,
    BSEDict=BSEDict)
```

Listing 2.3 How to tell COSMIC to please evolve a binary

Evolving multiple binaries The `Evolve` function can also take lists as arguments, meaning you can decide to evolve multiple binaries at a time. Don't go overboard either, especially if you decide to use the feature outlined in the next paragraph. This is a great tool to test what small differences in initial conditions can result in.

Plots and stuff A great feature is the `evolve_and_plot()` function that takes as input an Initial Binary table to produce an image file for each binary in that `initC` dataframe.

Note that I did not set any time limits to start with, but you should try to reframe the time range to better display the interesting evolutionary period in a binary's life.

```
fig = evolve_and_plot(initC, t_min=None, t_max=None, BSEDict=BSEDict,
    sys_obs={})
```

Listing 2.4 How to use COSMIC to plot a binary's evolutionary journey

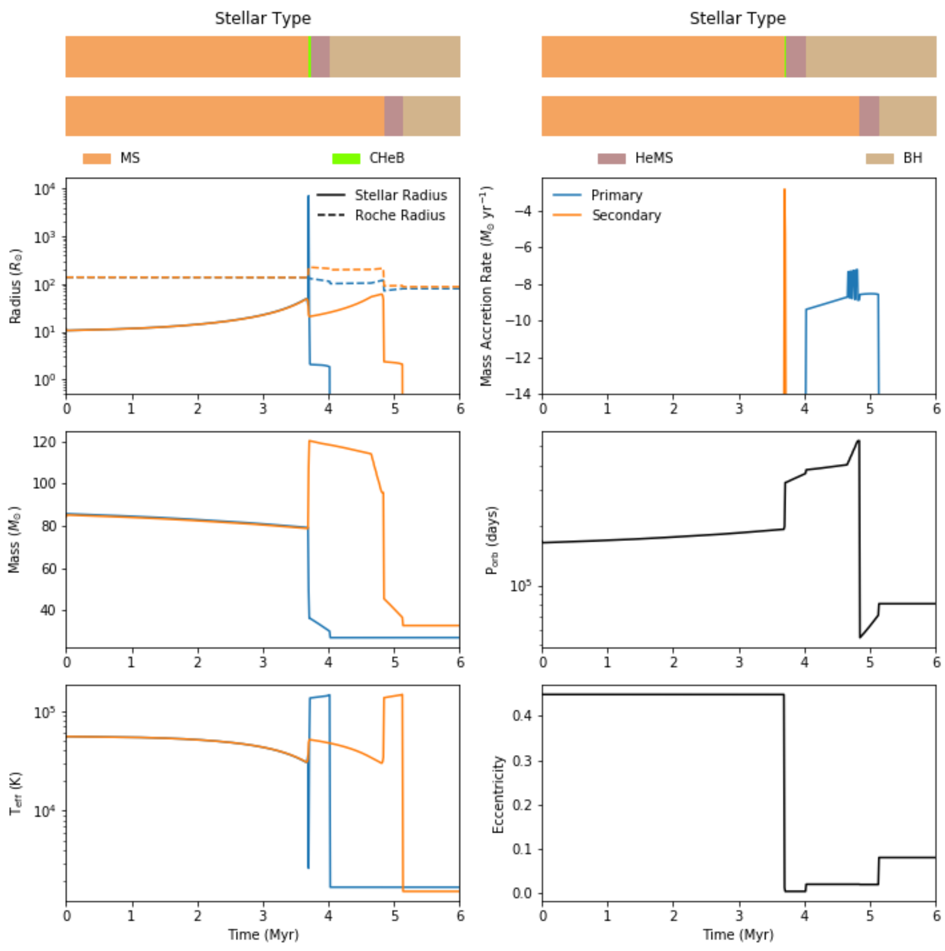


Figure 2.1 Example of a set of evolutionary plots produced by COSMIC, with a limits on the time range display `t_min=0, t_max=6` to focus on the [0, 6.0 Myr] period.

What's going on here? This feature might be helpful in getting a clearer picture of what happened to your binary over the course of its evolution.

For example, you can see from the plots above that the primary evolved from a main sequence star `[MS]`, to a Naked Helium Star `[HeMS]` after a brief Core Helium Burning period `[CHeB]`, to a black hole `[BH]` right before the 4 Myr mark. This coincided with a rapid circularization of the orbit. Similarly, the mass accretion onto the secondary star spiked – which is also visible in the simultaneous but opposite changes in mass of both primary and secondary star.

2.4 Scaling the fixed population to an astrophysical one [?]

Fixed population vs. astrophysical population The fixed population is effectively an histogram of binary properties given user-specified SFH and BSE models. Hence, it is to be distinguished from an actual, astrophysical population. The number of binaries of any type inside of the synthetic fixed population is **not** representative of anything other than how many binaries `cosmic-pop` had to generate and evolve in order to finally converge. Therefore, that number can and will change given different initial conditions.

Sampling Sampling works as follows:

How frequently a given type of star occurs should be the same in the fixed population and an astrophysical one. Meaning that:

$$\begin{aligned} \frac{N_{\text{astro, }\star}}{N_{\text{astro, tot}}} &= \frac{N_{\text{fixed, }\star}}{N_{\text{fixed, tot}}} \\ N_{\text{astro, }\star} &= N_{\text{astro, tot}} \times \frac{N_{\text{fixed, }\star}}{N_{\text{fixed, tot}}} \end{aligned} \tag{2.1}$$

where $N_\star$ denotes the number of a given type of star, say low-eccentricity white dwarfs, as opposed to N_{tot}, which refers to the total number of all types of stars within the population.

This works in a similar fashion if we use the total stellar mass of a population instead of the total number of systems in this population. In which case, equation 2.1 becomes:

$$N_{\text{astro, }\star} = N_{\text{astro, tot}} \times \frac{M_{\text{fixed, }\star}}{M_{\text{fixed, tot}}} \tag{2.2}$$

In both case, numbers corresponding to the fixed population are provided in the output of `cosmic-pop`, but the total number of systems in the astrophysical population or its total mass have to be entered by the user. However, if you're working on a Milky Way population, COSMIC has built-in models relying on McMillan+2011 [4] that carry this information so you don't have to.

```python
from cosmic import MC_samp
total_mass = max(np.array(total_mass))[0]
total_number_stars = max(np.array(N_stars))[0]
N_10_14_10_14_ThinDisk = MC_samp.mass_weighted_number(dat=conv,
    total_sampled_mass=total_mass, component_mass=MC_samp.
    select_component_mass('ThinDisk'))
thin_disk_sample = conv.sample(n=N_10_14_10_14_ThinDisk, replace=True)
```

Listing 2.5 Sampling a thin disk Milky Way astrophysical population from the synthetic fixed population

Assigning Star Formation History (i.e. birth times) The next step in transforming that statistical fixed population into an astrophysical one is to anchor each binary to a concrete Star Formation History by assigning them birth times.

Now, in case you have set `DeltaBurst` as your chosen SFH in the `Params.ini` file – which corresponds to a blank slate, all stars being born 13.7 Gyrs ago, and evolved to this day, you can *retroactively fit any SFH you want!* This is useful if you do not wish to run multiple fixed populations with the same BSE prescriptions, with only the SFH changing.

To do this, the simplest thing is to generate random ages within the agreed-upon star forming period of time – but more complex functions taking into account non-uniform Star Formation Rates can also be designed and applied by the user.

For the Thin Disk, following the information relayed in section 3.1.2, that period of time is assigned to be 0 to 10,000 Myr ago. Conversely, the Thick Disk is said to have been in active star formation 10,000 to 11,000 Myr ago and the Bulge 9,000 to 10,000 Myr ago.

We then logically filter out the binaries for which `tbirth` + `tphys` (the evolution time) surpasses the age of the Galaxy component.

```
thin_disk_sample['tbirth'] = np.random.uniform(0, 10000,
    N_10_14_10_14_ThinDisk)
thin_disk_sample = thin_disk_sample.loc[thin_disk_sample.tbirth +
    thin_disk_sample.tphys < 10000]
```

Listing 2.6 Assigning birth times to a sampled astrophysical population

Assigning galactic coordinates: the McMillan model Our binaries now have most parameters locked down, save for an actual localization. You can use a module already present within the COSMIC toolbox to take care of that, following the model outlined in McMillan+2011 [4] and his mass distribution functions.

This module, found in the `MC_samp.py` script of the COSMIC GitHub, can be adapted to a different model of the user's choosing, where parameters such as the Disk scale height can be modified or to fit geometric distributions for populations outside the Milky Way galaxy.

McMillan+2011 presents a Bayesian approach to fitting a parametrized mass model of the Milky Way to observational constraints. COSMIC uses the mass density models for each galaxy components described in the paper to assign galactic positions to the sampled binaries.

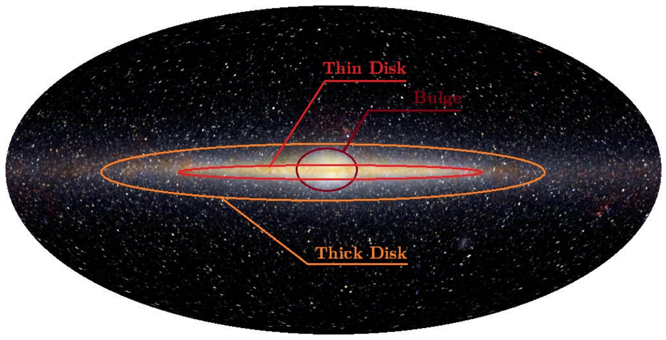


Figure 2.2 The McMillan model has geometric mass distributions for 3 of the main galaxy components: the Bulge, the Thin Disk, and the Thick Disk.

The given coordinates are in kpc, and are galacto-centric. You can compute the distance to Earth with a simple :

$$d = \sqrt{(x_\oplus - x_\star)^2 + (y_\oplus - y_\star)^2 + (z_\oplus - z_\star)^2} \tag{2.3}$$
$$\text{with } (x,\ y,\ z)_\oplus = (8.0,\ 0.0,\ 0.025) \tag{2.4}$$

The bulge is described as being azimuthally symmetric, with its radial and vertical density profile as follows:

$$\rho(r') \propto \frac{e^{-(r/r_{\text{cut}})^2}}{(1 + r'/r_0)^\alpha} \quad \text{where } r' = \sqrt{r^2 + (z/q)^2},\ \alpha = 1.8,\quad r_0 = 0.075 \text{ kpc},\ r_{\text{cut}} = 2.1 \text{ kpc, and } q = 0.5 \tag{2.5}$$

More over, following McMillan+2011, the total mass of the Bulge is assumed to be $M_{\text{bulge}} = 8.9 \times 10^9 M_\odot$, $\pm 10\%$.

Conversely, the Thin and Thick Disk are described as having similar distributions, with differing scale heights. They are also both assumed to be azimuthally symmetric, and distributed radially and vertically as a double exponential, as follows:

$$\rho(r)\rho(z) \propto e^{-r/r_h}\ e^{-z/z_h} \quad \text{Thin Disk}: \ r_h = 2.9 \text{ kpc},\ z_h = 0.3 \text{ kpc} \quad \text{Thick Disk}: \ r_h = 3.31 \text{ kpc},\ z_h = 0.9 \text{ kpc} \tag{2.6}$$

The masses of those components are $M_{\text{thin}} = 4.32 \times 10^{10} M_\odot$ and $M_{\text{thick}} = 1.44 \times 10^{10} M_\odot$. Note that those last 2 numbers are taken from the COSMIC 2019 paper, and I wasn't able to trace their origin back to the McMillan+2011 models.

```
xGx, yGx, zGx, inc, omega, OMEGA = MC_samp.galactic_positions(
   gx_component = 'ThinDisk',
   model        = 'McMillan',
   size         = len(thin_disk_sample))
thin_disk_sample['xGx']   = xGx
thin_disk_sample['yGx']   = yGx
thin_disk_sample['zGx']   = zGx
thin_disk_sample['inc']   = inc
thin_disk_sample['omega'] = omega
thin_disk_sample['OMEGA'] = OMEGA
```

Listing 2.7 Assigning galactic coordinates and orientations to a sampled astrophysical population

2.4.1 Gravitational Wave Evolution

The following goes over what the `GW_evol` function in the `MW_maker.py` script is all about. This is not downloaded as a part of COSMIC, but can be found on its Github. It is, however, helpful to understand what drives the rest of the evolution process of a binary following the formation of stellar remnants. Double degenerate binaries are mostly governed by orbital mechanics through the emission of gravitational waves, which circularize and shrink the orbit over time.

What follows allows us to compute a_{final} and e_{final} after t_{gw}, which is the time left between the formation of the stellar remnants and the present.

We start by computing t_{merge}, as if it is less than t_{gw}, then the system will have merged before the present, and we can remove it from our data set.

$$t_{merge} = \frac{a_0^4}{4\beta} \tag{2.7}$$

where the constant β is defined as $\beta = \frac{64}{5}c^{-5}G^3 m_1 m_2 (m_1 + m_2)$ and a_0 is the initial orbital separation.

A binary can be either **circular** or **eccentric**, and the category to which it belongs will determine how it will evolve over the course of its remaining evolution time t_{gw}.

For eccentric systems, for which $e_0 > 0.01$:

You feed in a_0 and e_0, which are the initial orbital separations and eccentricities respectively. From there, you can compute the corresponding constant c_0 as follows:

$$c_0 = a_0 \left(1 - e_0^2\right) e_0^{-12/19} \left(1 + \frac{121}{304} e_0^2\right)^{-870/2299} \tag{2.8}$$

Now, you divide the gravitational evolution time into 100 bins, in increments of $\delta t = t_n - t_{n-1}$. Now, for t_n in $[0, \frac{t_{gw}}{100}, 2\frac{t_{gw}}{100}, ..., t_{gw}] = [0, 1, 2, ..., 100]\delta t$, you can compute iteratively:

$$\begin{aligned} e_{n+1} &= e_n + \Delta e \\ &= e_n + \frac{de}{dt} \cdot \delta t \end{aligned}$$

where

$$\left\langle \frac{de}{dt} \right\rangle = -\frac{19}{12} \frac{\beta}{c_0^4} \frac{e^{-29/19} \left(1 - e^2\right)^{3/2}}{\left(1 + \frac{121}{304} e^2\right)^{1181/2299}} \tag{2.9}$$

$$\begin{aligned} a_{n+1} &= \frac{c_n e_{n+1}^{12/19}}{1 - e_{n+1}^2} \left(1 + \frac{121}{304} e_{n+1}^2\right)^{870/2299} \\ c_{n+1} &= a_{n+1} \left(1 - e_{n+1}^2\right) e_{n+1}^{-12/19} \left(1 + \frac{121}{304} e_{n+1}^2\right)^{-870/2299} \end{aligned} \tag{2.10}$$

This algorithm is iterated over n, until $n = 100$ and $a(t_{gw}) = a_{100} = a_{final}$.

For circular systems, for which $e_0 < 0.01$, which is approximated to $e_0 = 0$:

The computations are a lot more straightforward:

$$a_{final} = \left(a_0^4 - 4\beta t_{gw}\right)^{1/4} \tag{2.11}$$

For both, Kepler's III^rd law allows us to compute the final orbital period from m_1, m_2 and a_{final} as:

$$p_{orb} = \sqrt{\frac{a_{final}^3}{m_1 + m_2}} \tag{2.12}$$

3 Understanding COSMIC

3.1 Understanding the theory

This first section will give you some tools to understand some of the main concepts you are bound to encounter when you start using COSMIC – and more specifically, how they play into the grander scheme of stuff you can do with COSMIC.

3.1.1 Metallicity

Stars are made of stuff. In particular, we divide star stuff into fractional bits as:

$$\star = X + Y + Z = 1 \tag{3.1}$$

where X is the fraction of hydrogen the star is composed of, Y is helium, and Z is the metallicity, or "everything else." The metallicity of the Sun is $Z_\odot = 0.02$.

It is often given in units of "dex" [decimal exponent] as [Fe/H] $\equiv \log_{10}\frac{Fe/H}{[Fe/H]_\odot}$. For example:

$$\begin{aligned} \text{dex} &= -1 & \longrightarrow \quad & Z = 10^{-1} Z_\odot = 0.002 \\ &= 0 & \longrightarrow \quad & Z = 10^{0} Z_\odot = 0.02 \end{aligned}$$

Population of nearby stars are often born from the same gas and dust clouds, making metallicity a population parameter more than a single star parameter.

A thing of note is that supernovae are the major source of heavier elements in the Universe, typically. Hence, it's relatively safe to assume that older populations would have formed from gas poor in SNe-produced heavier elements, as the older a population is, the less time is left for SNe to occur before the stars form. Conversely, stars forming now will be able to contract gas that has already been polluted by SNe, and hence can be expected to be more metal-rich.

Note that halo stars have the lowest metallicity of all Galactic populations ([Fe/H] ~ -5), and hence might have been some of the first ones to form in the Milky Way!

3.1.2 Star Formation History (SFH)

Definition Star formation history is a global property of any population. It describes its periods of active star formation, and can include other parameters such as star formation rate (SFR, usually given in [$M_\odot \cdot \mathrm{yr}^{-1}$]), age-metallicity relation (AMR) or Initial Mass function (IMF, see section 3.1.4).

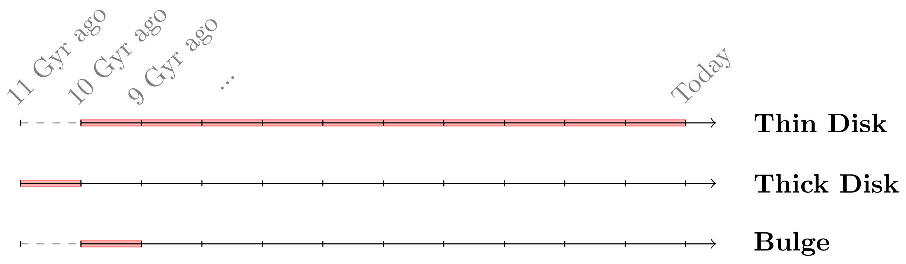


Figure 3.1 The SFH assumptions for the 3 main galactic components. In red, the periods of active star formation.

In COSMIC, assigning a given SFH to our population simply means giving a time of birth to each binary that lies within the corresponding period of active star formation.

Usually, the choice in `Params.ini` of what galaxy component to sample and evolve informs what pre-set SFH model to use. Those definitions are embedded into the COSMIC code itself.

Galaxy Component	SFH model	Component Age (Myr)
Thin Disk	`'const'`	10000.0
Bulge	`'burst'`	10000.0
Thick Disk	`'burst'`	11000.0
Delta Burst	`'delta_burst'`	13700.0

- `'const'` assigns an evolution time assuming a constant star formation rate over the age of the Milky Way disk. By default, this means giving binaries birth times that range from 0 to 10 Gyr ago, and evolving them until today (0 years ago).
- `'burst'` assigns an evolution time assuming a burst of constant star formation for 1 Gyr starting at `component_age` in the past.
- `'delta_burst'` assigns all binaries the same $t = 0$ birth time, and evolves them for the full component age, which is by default 13.7 Gyr – or until the event that you have set as your trigger to compute convergence (*e. g.* 1st supernova, formation of your user-specified `kstar`, etc)

A strategic choice for running `cosmic-pop`, especially if your focus is on LISA populations, is to choose a `'delta_burst'` construction – which is stripping the process to its bare essentials to focus on the Binary Evolution part. You can then retroactively assign your chosen SFH parameters (equivalent to simply assigning birth times) to fit your model.

3.1.3 Binary Stellar Evolution (BSE)

In this context, BSE refers to the whole algorithm your binaries are subjected to in order to predict their end states from their early days. It is a highly complex set of models and rules that aims to translate what we currently know about how binaries live their lives into lines of code you can feed your computer to explore new pathways.

3.1.4 Initial Mass Function (IMF)

This is often grouped in with Star Formation History because it refers to the property of ZAMS stars that are going to be evolved – in other words, to star formation. In COSMIC, you can choose between 2 models to initialize your binary population by generating their initial masses, orbital periods, eccentricity, etc.

`independent` initializes binaries with parameters distributions independent of each other for the primary mass, mass ration, eccentricity, separation and binary fraction (percentage of stars that are born as part of a binary).

`multidim` recognizes correlations between those parameters and initializes binaries with multidimensional parameter distributions according to Moe & Di Stefano+2017 [5].

3.1.5 Signal-to-Noise Ratio (SNR)

While only tangential to population synthesis, SNR is inevitable in discussions of gravitational wave signals. When generating a binary population, you might want to know "are any of those going to emit strong enough gravitational radiation to be observable?"

The SNR provides you with a very simple way to evaluate that. The signal to noise ratio is, as its name might suggest, the ratio of the strength of your signal over your detector's sensitivity.

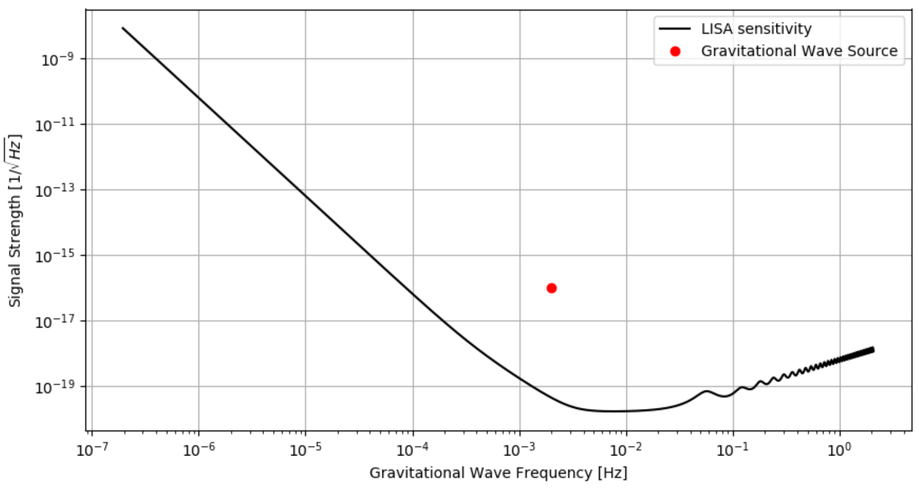


Figure 3.2 LISA's inherent sensitivity plotted next to a given signal (in red) gives a quick visual interpretation of the SNR. LISA cannot resolve any source below the black line. Conversely, the signal in red is shown to sit comfortably above the line, with a SNR of over 100 – a very strong source!

3.2 Understanding the inputs

3.2.1 The `Params.ini` file

The unavoidable input file – its format looks like a remnant of the legacy Fortran BSE code [3] upon which COSMIC is built.

Table 3.1 provides a brief overview of what's in the file. However, some choices are far from inconsequential, and their implications are detailed in the paragraphs to follow.

Section	Details
`Filters`	What to retain in the final output files? Only the final state of each binary, or also states mid-evolution?
	Also, do I only want to keep binaries who at the end of the evolution time are still untouched or am I fine with keeping some that have merged or have been disrupted in some way?
`Sampling`	Sampling of initial Zero Age Main Sequence population
	Galaxy component (Thin Disk, Bulge, etc) to sample – informs SFH
	Metallicity
`Convergence parameters`	What are the parameters I want to see converging? Mass, eccentricity, orbital period, etc ... ?
	At what evolutionary stage are we checking for that convergence?
	Should we filter out from our `bcm`, `bpp` and `initCond` dataframe the binaries that didn't make the cut into our `conv` dataframe?
	What's my convergence tolerance? How close do the parameters have to be for me to accept them?
`Random seed`	What random seed am I using for this `cosmic` run?
`BSE models`	Sampling (set length of time steps for the `bcm` dataframe)
	Wind (choose model for wind mass loss & set additional wind parameters)
	Common envelope
	Kick flags
	Remnant mass
	Remnant spin
	Mass transfer
	Tides
	White dwarf
	Pulsar
	Mixing variables
	Magnetic braking
	Convective vs. radiative boundaries

Table 3.1 What you typically find in the `Params.ini` file

A more in-depth overview: You will find below a breakdown of some of the main modifiable parameters in `Params.ini`. This is not an exhaustive list (namely, most BSE parameters aren't touched upon), but should be complete enough to enable a solid understanding of the options laid out in front of you.

`[Filters]`

```
; select_final_state will retain only the final entry of the bcm arry if set
   to True
select_final_state = True
```

As mentioned in section 3.3.3, the `bcm` dataframe can store the state of the binary at user-defined time steps, or simply hold onto the very last entry – the final state of each evolved binary. If this is set to `True`, you will only keep the final states in the `bcm` array.

```
; binary_state determines which types of binaries endstates to retain
; 0 alive today, 1 merged, 2 disrupted
binary_state = [0,1,2]
```

In the course of their evolution, binaries might merge or get disrupted by something. This line of code asks if you'd like to still keep binaries that might indeed not be alive at the very end of the evolution time.

`[Sampling]`

```
; Specify if you woud like to sample initial conditions via
; the independent method (independent) or would like to sample
; initial conditions follow Moe & Di Stefano (2017) (multidim)
sampling_method=multidim
```

This specifies what method to use to dictate the Initial Mass Function (IMF) from which your ZAMS binaries will draw from as they're randomly generated. In short, the `independent` method has independent distributions for each parameter, while the `multidim` method accounts for the correlation between initial parameters with a multi-dimensional distribution.

```
; Galaxy Components. Options include Bulge, ThinDisk, ThickDisk, and
   DeltaBurst
; Think Star Formation History with which to use as your basis for sampling
galaxy_component = Delta_Burst
```

This choice informs what default Star Formation History to automatically assign to your population. The safest choice regardless of what galaxy component you'd like to sample, is to choose '`Delta_Burst`', which assigns the same birth times to all stars, allowing you to retroactively fit to your population your customized Star Formation History later on in the process. This allows the `cosmic-pop` run to fully focus on the Binary Evolution part of the process. See section 3.1.2 for more details.

```
; Metallicity of the population of initial binaries
metallicity = 0.02
```

The agreement is that the stellar population in the Thin Disk and the Bulge follow solar metallicity $Z_{\text{Bulge}} = Z_{\text{Thin Disk}} = Z_\odot = 0.02$, while Thick Disk populations have metallicity 15% that of the Sun, meaning $Z_{\text{Thick Disk}} = 0.15\, Z_\odot = 0.003$. See section 3.1.1 for a more in-depth account of what metallicity's all about.

```
[Convergence Parameters]
```

```
; A list of parameters you would like to verify have converged
; to a single distribution shape.
; Options include mass_1, mass_2, sep, porb, ecc, massc_1, massc_2
; rad_1, rad_2
convergence_params = [mass_1,mass_2,porb,ecc]
```

Those are simply the parameters you want to see have converged. The standard choices of parameters to specify would be the masses of the two stars and their orbital period. Eccentricity is only interesting for systems with either a neutron star or a black hole, which present a wide range of eccentricities, as opposed to double white dwarfs, which are most often quite circular.

```
; convergence_limits is a dictionary that can contain limits for convergence
    params
; convergence_limits = {"mass_1" : [0, 20], "sep" : [0,5000]}
convergence_limits = {}
```

Among the parameters you have specified above, there are maybe some that you would like to place some limits upon. Note that if you are placing any one limit, you have to specify a corresponding upper or lower limit as well.

```
; formation computes convergence on binary properties
; at formation with user-specified final kstars

; 1_SN computes convergence on binary properties
; just before the first supernova for the population with
; user-specified final kstars

; 2_SN computes convergence on binary properties
; just before the second supernova for the population with
; user-specified final kstars

; disruption computes convergence on binary properties
; just before disruption of the population with
; user-specified final kstars

; final_state computes convergence on binary properties
; after the full evolution specified by the user-supplied evolution time
; and with the user specified final kstars

; XRB_form computes convergence on binary properties
; at the start of RLO following the first supernova on the population with
; user-specified final kstars
convergence_filter = formation
```

This parameter is less innocuous than it may look. If you are interested in compact objects (`kstar` anywhere from `10` to `14`), a judicious choice is to pick `formation` as the step at which to compute convergence. It all depends on what aspect of your binary population you are interested in looking at.

```
; match provides the tolerance for the convergence calculation
; and is calculated as match = log10(1-convergence)
; default = -5.0
match = -6.0
```

This determines how stringent you would like the convergence determination to be. How convergence is computed is laid out in detail in section 2.1.2 of Breivik+2019 [2].

`[Random Seed]`

```
; random seed int
seed = 42
```

This sets the random seed used for sampling from the Initial Mass Function (IMF) to generate your ZAMS stars and later evolve them. It is important for reproducibility. This is particularly useful when comparing two populations, e.g. binary black holes and binary neutron stars, which should be simulated separately, but using the same `rand_seed` value.

`[BSE]`

```
; qcflag is an integer flag that sets the model to determine which critical
    mass ratios to use for the onset of unstable mass transfer and/or a common
    envelope. NOTE: this is overridden by qcrit_array if any of the values are
    non-zero.
; 0: standard BSE
; 1: BSE but with Hjellming & Webbink, 1987, ApJ, 318, 794 GB/AGB stars
; 2: following binary_c from Claeys+2014 Table 2
; 3: following binary_c from Claeys+2014 Table 2 but with Hjellming & Webbink,
     1987, ApJ, 318, 794 GB/AGB stars
; 4: following StarTrack from Belczynski+2008 Section 5.1. WD donors follow
    standard BSE
qcflag=4
```

This parameter determines what critical mass ratios to use for the onset of unstable mass transfers and/or common envelope. This factors in with systems consisting of a donor accreting onto a compact companion and determines whether or not a binary is stable or not. Different models have been shown to produce discernibly different results, hence the inclusion of `qcflag` in this guide.

3.2.2 What does `kstar` refer to?

Different integer values for `kstar` refer to different stellar evolutionary states, as described in table 3.2. This follows the conventions established in Hurley+2001's BSE code [3].

`kstar`	**Evolutionary state**
0	Main Sequence (MS) < 0.7 [$M_\odot$]
1	MS > 0.7 [$M_\odot$]
2	Hertzsprung Gap
3	First Giant Branch
4	Core Helium Burning
5	Early Asymptotic Giant Branch (AGB)
6	Thermally Pulsing AGB
7	Naked Helium Star MS
8	Naked Helium Star Hertzsprung Gap
9	Naked Helium Star Giant Branch
10	Helium White Dwarf
11	Carbon/Oxygen White Dwarf
12	Oxygen/Neon White Dwarf
13	Neutron Star
14	Black Hole
15	Massless Remnant

Table 3.2 Evolutionary State `kstar` of the Star

3.3 Understanding the outputs

Running `cosmic-pop` will output: (1) a `log` text file and (2) an `hdf5` file which contains several `DataFrames`. This will inevitably contain more information that you should need, but in case you ever need to check for errors, you can find everything in there.
Those different dataframes are accessed in the following way, where the `key` parameter should be substituted by any of the terms you can find below:

```
conv = pd.read_hdf(myfile, key='conv')
```

Listing 3.1 How to extract data from the output file

1. `conv`
   The converged population whose parameters are specified by the convergence subsection of the inifile and input `kstar` values. See section 3.3.2
2. `bpp`
   The evolutionary history of binaries which satisfy the user-specified final kstars and filter in the convergence subsection. See section 3.3.1.
3. `bcm`
   The final state of binaries in the bcm array which satisfy the user-specified final kstars and filter in the convergence subsection. See section 3.3.3.
4. `initCond`
   The initial conditions for each binary which satisfies the user-specified final kstars and filter in the convergence subsection
5. `idx`
   An integer that keeps track of the total number of simulated binaries to maintain proper indexing across several runs of cosmic-pop
6. `match`
   Tracks the convergence where `match = log10(1-convergence)`
7. `mass_binaries`
   Tracks the total mass of binaries needed to create the fixed population
8. `mass_singles`
   Tracks the total mass of single stars needed to create the fixed population; if the binary fraction is 100%, the mass in singles will be zero
9. `mass_stars`
   Tracks the total mass of all stars, including binaries and singles, that were needed to create the fixed population
10. `n_binaries`
    Tracks the total number of binaries that were needed to create the fixed population
11. `n_singles`
    Tracks the total number of single stars needed to create the fixed population
12. `n_stars`
    Tracks the total number of stars, where `n_stars = n_singles + 2*n_binaries`, needed to create the fixed population

3.3.1 The bpp dataframe

The bpp dataframe records a selection of binary parameters at key evolutionary changes, such as kstar change, contact, binary disruption or supernova of either primary or secondary. It is the evolutionary history of each binary.

Variables with the subscript "_1" are referring to the primary star. As you can expect, columns names with a "_2" ending are referring to the secondary star.

Parameter	Name	Units
tphys	Evolution time	[Myr]
mass_1	Primary mass	[$M_\odot$]
kstar_1	Primary evolutionary state	
sep	Semimajor axis	[$R_\odot$]
porb	Orbital period	[days]
RROL_1	Primary radius divided by Roche lobe radius	
evol_type	Type of key moments in evolution	
Vsys_1	Change in systemic velocity due to first SN	[km/s]
SNkick	Magnitude of SN natal kick	[km/s]
SNtheta	Angular change in orbital plane due to SN	[degrees]
aj_1	Effective age of the primary	[Myr]
tms_1	Primary main sequence lifetime	[Myr]
massc_1	Primary core mass	[$M_\odot$]
rad_1	Primary radius	[$R_\odot$]
bin_num	Unique binary index that is consistent across initial conditions, bcm and bpp DataFrames	

Table 3.3 Contents of the bpp dataframe

3.3.2 The conv dataframe

The conv dataframe records the exact same parameters as the bpp dataframe (see above section 3.3.1 for a breakdown of the columns), but records only the binary states when convergence is computed.

It only contains the binaries whose parameters fit the convergence parameters you set in the Params.ini file and on the command line – meaning only stars with the kstar numbers and convergence limits you asked for.

3.3.3 The `bcm` dataframe

The `bcm` dataframe is a slightly different selection of binary parameters at user-specified timesteps in the evolution or only their final states, depending on the `Params.ini` file.

Parameter	Name	Units
`tphys`	Evolution time	[Myr]
`kstar_1`	Evolutionary state	section 3.2.2
`mass0_1`	Previous evolutionary stage primary mass	[$M_\odot$]
`mass_1`	Primary mass	[$M_\odot$]
`lumin_1`	Primary luminosity	[$L_\odot$]
`rad_1`	Primary radius	[$R_\odot$]
`massc_1`	Primary core mass	[$M_\odot$]
`radc_1`	Primary core radius	[$R_\odot$]
`menv_1`	Primary envelope mass	[$M_\odot$]
`renv_1`	Primary envelope radius	[$R_\odot$]
`epoch_1`	Primary epoch	[Myr]
`spin_1`	Primary spin	[rad/yr]
`deltam_1`	Primary mass transfer rate	[$M_\odot$/yr]
`RROL_1`	Primary radius divided by Roche lobe radius	
`porb`	Orbital period	[days]
`sep`	Semimajor axis	[$R_\odot$]
`ecc`	Eccentricity	
`B_0_1`	Initial neutron star magnetic field	[G]
`SNkick_1`	Magnitude of first SN natal kick	[km/s]
`Vsys_final`	Final systemic velocity magnitude	[km/s]
`SNtheta_final`	Angular change in orbital plane due to SN	[degrees]
`SN_1`	Supernova type: `[1]` : Fe Core-collapse SN `[2]` : Electron capture SN `[3]` : Ultra-stripped supernovae `[4]` : Accretion induced collapse SN `[5]` : Merger induced collapse `[6]` : Pulsational-pair instability `[7]` : Pair instability SN	
`bin_state`	State of the binary: `[0]` : binary `[1]` : merged `[2]` : disrupted	
`merger_type`	String of the kstar's in the merger, `-001` if not merged	
`bin_num`	Unique binary index that is consistent across all DataFrames	

Table 3.4 Contents of the `bcm` datafram

A Running COSMIC on Quest

I am using FastX to make a network connection to Quest through SSH. This allows me to access the Quest Linux servers with full graphics support.

How to install FastX and set up an SSH connection to Quest is already explained online, so I won't go into details. Instead, I will try to provide the COSMIC specific instructions you won't be able to find as easily.

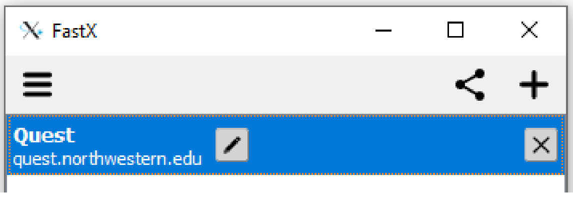


Figure A.1 a FastX window

A.1 Initializing COSMIC for the first time

To initialize a COSMIC environment specifically on Quest, you'll first need to load the built-in Anaconda module, so that you can use `conda` to install COSMIC. If like me, you like to use the interactive `IPython` environment or Jupyter notebooks, note that you will have to install them *inside* of your COSMIC environment as follows.

```
module load anaconda3
conda create --name cosmic python==3.7 numpy h5py
source activate cosmic
pip install --upgrade cosmic-popsynth
pip install ipython jupyter
```

Listing A.1 Run this to install and initiate COSMIC on Quest

A.2 Running COSMIC

By simply logging onto Quest, you are immediately directed to your virtual home directory where you can store some files and perform simple tasks. However, both the computing power and disk space you're allotted there are limited. Alongside simple access to Quest, you should have been added to CIERA allocations, such as `b1095`, which is the allocation dedicated to gravitational wave research. This is your pass to asking for more power.

For additional disk storage, you are encouraged to make a folder inside of the projects directory and treat that as your home folder. This folder is a shared resource but has far more storage – up to 70,000 GB, or 70 TB, for `/projects/b1095`. Comparatively, your original home directory at `/home/<NetID>` has a limit of about 80 GB.

To have access to more RAM (which is why we're using Quest in the first place), you have to request additional memory for a temporary time by submitting what's called "jobs". There are different kinds of jobs that are best suited to different purposes.

Interactive Jobs Here, the idea of "job" can be slightly misleading, as what this is doing is launching an interactive session that, in terms of interface, is not that different from what you had at your disposition before – except that you now have access to a lot more memory, with full graphics support.

```
srun --x11 --account=b1095 --time=00:30:00 --partition=grail-std -N 1 -
   n 1 --mem=5G --pty bash -l
```

Listing A.2 Sample command line to submit an interactive job

In short, `--x11` allows for your computer to display graphics coming from your Quest session, like a Python plot. This is also what you'd choose if you wanted to run a Jupyter notebook. More info on that specifically is in the next section.

Batch Jobs When you submit a batch job, you send Quest a list of instructions to follow. You can't interact with it, and can't change the job instructions until it is completed (or has failed). It's the perfect infrastructure for running something like `cosmic-pop`. I named the following file `cosmic-pop.sh`:

```
#!/bin/bash
#SBATCH -A b1095                # Allocation
#SBATCH -p grail-std            # Queue
#SBATCH -t 12:00:00             # Walltime/duration of the job
#SBATCH -N 1                    # Number of Nodes
#SBATCH -n 1                    # Number of Cores (processors)
#SBATCH --mem=1G                # Memory per node in GB needed
#SBATCH --mail-user=jlm@u.northwestern.edu  # Your email
#SBATCH --mail-type=END,FAIL    # Emails when your job ends or fails
#SBATCH --output=bulge.out      # Output log for possible errors
#SBATCH --job-name="bulge"      # Nickname of job

export PATH=$PATH:/projects/b1095/jlmalewicz/cosmic/001-cosmic-pop/
module purge all
module load anaconda3

source activate cosmic
cosmic-pop --final-kstar1 10 14 --final-kstar2 10 14 --inifile
   Params_Bulge.ini --Nstep 5000 --Niter 10000000 -n 1
```

Listing A.3 Example of a script for a batch job

Now, submitting it is as easy as typing in `sbatch cosmic-pop.sh` and pressing `"Enter"`!

Optional: Parallelized Batch Jobs In short, the architecture of the CIERA Quest allocation consists of 37 nodes, each made up of 28 cores, or processors. COSMIC can be parallelized within nodes, but not across. To speed up the `cosmic-pop` process, you can modify the submission batch script to instruct COSMIC to use more than 1 node.

```
#SBATCH -n 10
#SBATCH --mem-per-cpu=2G
cosmic-pop --final-kstar1 10 14 --final-kstar2 10 14 --inifile
   Params_Bulge.ini --Nstep 5000 --Niter 10000000 -n 10
```

A.3 Miscellaneous computer things

Install COSMIC from the `develop` branch First, you will need a GitHub account. Once you have that, move to the directory where you want to store the source code, then you can clone the COSMIC repository onto your computer by entering:

```
git clone https://github.com/COSMIC-PopSynth/COSMIC.git
```

Once that's done, `cd` into the COSMIC folder then type '`pip install .`'

How to convert your files from `hdf` to `csv`: Don't panic, just run the following, substituting the `'gx_dat...'` string with any other filename:

```python
import pandas as pd
gx = pd.read_hdf('gx_dat_BH_BH_ThinDisk.h5', key='galaxy')
gx.to_csv('gx_dat_BH_BH_ThinDisk.csv')
```

Listing A.4 Sample python script to convert standard COSMIC output into human-readable `csv` format.

The `key` tells python sub-file you want to extract from the `hdf` file. Here, the `key` is ‘`galaxy`‘, but for COSMIC fixed populations, the options are given in section 3.3.

Copying files between Quest and your local computer To transfer files to Quest from Windows, you need to start from a Windows terminal, and run the following:

```
pscp \<path>\<file> <NetID>@quest.it.northwestern.edu:/home/<NetID>/<
   path>/<file>
```

Listing A.5 Command line to copy files from a Windows machine to Quest

To copy from Quest to Windows, simply switch the arguments to put the path to the file you'd like to copy first.

Using Jupyter notebook You can run Jupyter on a high-performance computing node through an interactive job. More info at `https://kb.northwestern.edu/88448`.

```
module load anaconda3
source activate cosmic
jupyter notebook --no-browser
```

Listing A.6 Lines to run on Quest terminal

You might be assigned a port, which will be a number like `8899`, or similar. To make sure you aren't trying to connect to a port that might already be in use, you can also specify one yourself by adding `--port=6666` e.g. (any number between 5000 and 9999 should do) to the `jupyter notebook` line.

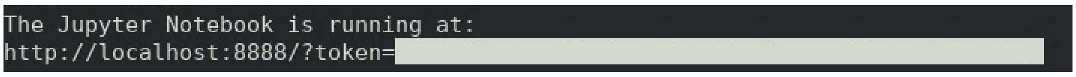
Write down your hostname as well. It's the node you were assigned by Quest – just run `hostname` to get it if you don't know where to find it.

Now, you can establish the `ssh` connection on your local computer by running the following on your computer terminal:

```
ssh -L 8899:localhost:8899 <your_netID>@quest.northwestern.edu ssh -N -
   L 8899:localhost:8899 <hostname>
```

Listing A.7 Lines to run on your local (Windows or else) terminal

Finally, just enter the address `http://localhost:8899/` on your browser or use the one given in the Quest terminal. You might be asked to enter a token, which can be found here:



```
The Jupyter Notebook is running at:
http://localhost:8888/?token=
```

Figure A.2 Snapshot of a Quest terminal used to access Jupyter notebook. Highlighted is where you would copy the token in question. Alternatively, you can just use the link itself as is given.

B Documenting my attempts at a fiducial LISA galaxy

B.1 Objectives

The goal is to simulate a fiducial population of galactic LISA sources, following the current agreement of the LISA consortium regarding binary stellar evolution models.

We start by defining the process with which we hope to simulate the most standard population of double degenerate binaries possible. This mostly follows the conventions chosen for the 2019 COSMIC paper [2].

How many populations should we run for best results? We will run a total of $2 \times 9 = 18$ fixed populations to account for the 2 standard galactic metallicities $Z = Z_{\odot} = 0.02$ and $Z = 15\%\ Z_{\odot} = 0.003$ and the 9 different possible combinations of stellar types – He + He, CO + He, CO + CO, ONe + WD, NS + WD, BH + WD, NS + NS, BH + NS, and BH + BH. We run them separately so as to force COSMIC to compute convergence on each sub-population separately, as they can have wildly different distributions between them.

B.2 Detailed process

Star Formation History The chosen galaxy component is "Delta_Burst", as this allows us to specify star formation history later on and allow COSMIC and its convergence calculator to focus exclusively on the Binary Evolution part of the process.

Convergence parameters For double white dwarf binaries, you can specify the following in the `Params.ini` file:

```
[white dwarf binaries]
convergence_params = [mass_1, mass_2, porb]
convergence_limits = {"sep" : [0, 1000]}
```

As opposed to the following if the population involves a black hole or neutron star component:

```
[binaries with a neutron star or a black hole]
convergence_params = [mass_1, mass_2, porb, ecc]
convergence_limits = {}
```

`["ecc"]` – Firstly, the double white dwarfs that evolve to the LISA band are almost always circular, hence it doesn't make sense to compute the convergence for the eccentricity distribution.

`["sep"]` – Secondly, the double white dwarf binaries with separations above 1000 $R_{\odot}$ won't evolve into the LISA band in a Hubble time, so there is also no need to worry about them.

`["porb"]` – Finally, note that not all sources will be in the LISA frequency band. However, with gravitational wave emission, we expect their orbital period to decrease, thus possibly entering the frequency range detectable by LISA. Hence, we do not place any restrictions on the orbital period at this stage.

Running the `cosmic-pop` executable Not only are you running 18 sub-populations in total, but the run time for each of those populations has increased. The convergence is computed on sub-populations of a sub-population of possible COSMIC outputs.

For that particular endeavor, I wrote a script that you would submit once for each metallicity, which loops over the contents of a file named `params.txt` that you can find below.

```
#!/bin/bash

while IFS=$'\t' read kstar1 kstar2 params stringkstar2
do
  JOB=`sbatch << EOJ
#!/bin/bash
#SBATCH -A b1095
#SBATCH -p grail-std
#SBATCH -t 72:00:00
#SBATCH -N 1
#SBATCH -n 14
#SBATCH --mem-per-cpu=1G
#SBATCH --mail-user=jlm@u.northwestern.edu
#SBATCH --mail-type=END,FAIL
#SBATCH --job-name="met02.${kstar1}.${stringkstar2}"
#SBATCH --output="./met02.${kstar1}.${stringkstar2}.out"

export PATH=$PATH:/projects/b1095/jlmalewicz/cosmic/001-cosmic-pop/feb
    -04/met-02
module purge all
module load anaconda3

source activate cosmic
cosmic-pop --final-kstar1 ${kstar1} --final-kstar2 ${kstar2} --inifile
    ${params} --Nstep 14000 --Niter 10000000 -n 14
EOJ
`

echo "$JOB for parameters kstar1=${kstar1} kstar2=${kstar2} submitted
    on `date`"
sleep 2s
done < params.txt
exit
```

Listing B.1 Sample script to submit in order to send 9 batch jobs to run each ***`cosmic-pop`*** *subpopulation. Here given for metallicity* $Z_\odot = 0.02$

```
10  10  Params_02_WD.ini  10
11  10  Params_02_WD.ini  10
11  11  Params_02_WD.ini  11
12  10 12 Params_02_WD.ini  10_12
13  10 12 Params_02_else.ini  10_12
14  10 12 Params_02_else.ini  10_12
13  13  Params_02_else.ini  13
14  13  Params_02_else.ini  13
14  14  Params_02_else.ini  14
```

Listing B.2 Sample ***`params.txt`*** *file that provides the input parameters for the script above to loop over. The parameters are tab-separated (\t)*

The following lines of code should be ran from the bash terminal in order to (1) make the all-cosmic-pop.sh file executable and (2) execute it.

```
chmod u+x all-cosmic-pop.sh
./all-cosmic-pop.sh
```

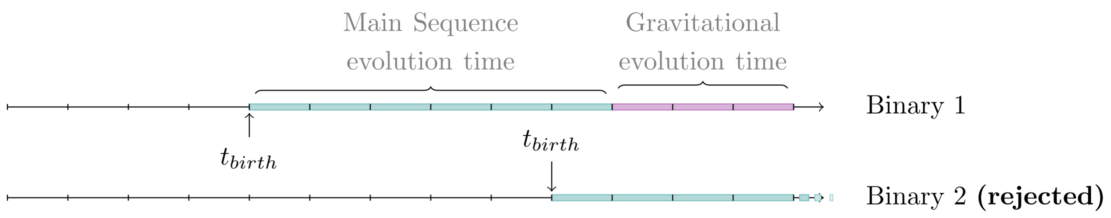
Sampling, assigning birth times and galactic positions We simulated a Delta Burst type population, which provided us with a blank slate in terms of star formation history, and computed convergence on our population right as they evolved into zero-age compact binaries. The tphys column therefore simply corresponds to the evolution time between their birth as ZAMS to the formation of the compact binaries.

We can now assign real birth times to each binary as being anytime during the star forming years of the galaxy component we are sampling. The fiducial choice is the one defined in section 3.1.2. We only keep binaries for which doesn't exceed the total age of the galactic component we are sampling.

Additionally, this is now the time to give them galactic positions as well, by following the McMillan+2011 model distributions, outlined in section 2.4.

The last stretch: Gravitational Wave emission Once the binaries evolve into white dwarfs, neutron stars and/or black holes, conventional BSE has little influence on the rest of their lifetime. What really defines the rest of their evolution is binary mechanics and gravitational wave emission.

Hence, with the time left between the formation of the binary and the present day, we subject it to standard gravitational evolution following the model codified in Peters & Mathews+1963 [7] and Peters+1964 [6] and explained in section 2.4.1.



Above is an qualitative example to understand how realistic timescales are assigned to each binary in your data set. The time of birth is randomly assigned, following the chosen SFH for the whole population. In blue, the main sequence evolution time (tphys) is the time between birth of the ZAMS binary and formation of the compact binary. In purple is the time left for gravitational evolution.

Binary 2 is shown to have been rejected by the algorithm because its time of birth + main sequence evolution time surpasses the age of the galaxy component we're sampling.

References

[1] K. Breivik and S. C. Coughlin. COSMIC Documentation. *https://cosmic-popsynth.github.io/*, 2019.

[2] K. Breivik, S. C. Coughlin, M. Zevin, C. L. Rodriguez, K. Kremer, C. S. Ye, J. J. Andrews, M. Kurkowski, M. C. Digman, S. L. Larson, and F. A. Rasio. *COSMIC Variance in Binary Population Synthesis.* Nov 2019.

[3] J. R. Hurley, C. A. Tout, and O. R. Pols. *Evolution of Binary Stars and the Effect of Tides on Binary Populations. Monthly Notices of the Royal Astronomical Society*, 329(4):897–928, Feb 2002.

[4] P. J. McMillan. *Mass Models of the Milky Way. Monthly Notices of the Royal Astronomical Society*, 414(3):2446–2457, Apr 2011.

[5] M. Moe and R. Di Stefano. *Mind Your Ps and Qs: The Interrelation between Period (P) and Mass-ratio (Q) Distributions of Binary Stars.* , 230(2):15, June 2017.

[6] P. C. Peters. *Gravitational Radiation and the Motion of Two Point Masses. Phys. Rev.*, 136:B1224–B1232, Nov 1964.

[7] P. C. Peters and J. Mathews. *Gravitational Radiation from Point Masses in a Keplerian Orbit. Physical Review*, 131(1):435–440, Jul 1963.