Build Initial Data

January 29, 2025

1 Introduction and Set Up

Based on Zhu et al. 2025, our goal is to derive the statistical characteristics of binary black hole systems in actuality. Therefore, we must conduct a simulation utilizing SEVN (Spera et al. 2018) with initial conditions derived from observational data within our own galaxy. This notebook is dedicated to generating the initial conditions for the SEVN simulation.

Zhu et al. 2025: https://doi.org/10.48550/arXiv.2409.14159
Spera et al. 2018: https://doi.org/10.1093/mnras/stz359

SEVN userguide can be found in: https://gitlab.com/sevncodes/sevn

To run the simulation, we need to provide input values for the mass of the primary and secondary stellar objects, their respective metallicities, dimensionless stellar spins, supernova types, initial stellar ages, semimajor axes, eccentricities, simulation end time, and the output time schedule.

[1]:
```
# Scale factor

DAY=3600 * 24   # second
G = 6.6743e-11  # m3 kg-1 s-2
```

[2]:
```
# Set number of datapoints for each kernal
number=60000000
```

2 Primary and Secondary Stellar Mass Distribution

The primary mass follows a power-law distribution based on Kroupa, 2001

$$\xi(M_1) \propto M_1^{-2.3}\,, \quad M_1 \in [10M_\odot, 150M_\odot]\,.$$

Kroupa, 2001: https://doi.org/10.1046/j.1365-8711.2001.04022.x

[16]:
```
# Define the power law code based on the distribution above

import numpy as np

def M1powerlaw(x, idx=-2.3, start=10,end=150):
    return x**idx

class SamplingProbability():
```

```
    def __init__(self, pdf, domain, isLog=False, **kwargs):
        self.pdf = pdf
        self.domain = domain
        self.isLog = isLog
        self.pdf_args = kwargs
        self._compute_cdf()
    def eval_pdf(self, x):
#         if self.isLog:
#             return np.log10(self.pdf( x, **self.pdf_args ))
        return self.pdf( x, **self.pdf_args )
    def _compute_cdf(self):
        self.generate_grid_points()
        p = self.eval_pdf(self.grids)
        if self.isLog:
            ln10 = np.log(10) # this normalization constant can drop.␣
 ↪__normalize will cancel it
            self.cdf = ln10 * np.cumsum( p*self.grids  )
        else:
            self.cdf = np.cumsum(p) # if it is log grid, should not use this␣
 ↪formula
        self.__normalize_cdf()
    def generate_grid_points(self, precision=100):
        if self.isLog:
            __lower = np.log10(self.domain[0])
            __upper = np.log10(self.domain[1])
            self.grids = 10**np.linspace(__lower, __upper, precision)
        else:
            self.grids = np.linspace( self.domain[0], self.domain[1], precision )
    def __normalize_cdf(self):
        self.cdf -= np.min(self.cdf)
        self.cdf /= np.max(self.cdf)
        # check cdf normalization
        assert np.max( self.cdf ) == 1 and np.min(self.cdf) == 0
    def generate_samples(self, n_samples):
        # for the explanation, see the wiki page of "inverse sampling transform"
        u = np.random.random(n_samples)
        # numerical inverse of cdf (monotonic) is just reversed grid points
        if self.isLog:
            lgu = np.log10(u)
            return 10**(np.interp( lgu, np.log10(self.cdf), np.log10(self.grids)␣
 ↪))
        else:
            return np.interp( u, self.cdf, self.grids )
```

[17]:
```
PM1=SamplingProbability(M1powerlaw, [10,150] )

# sample data points from M1
```

```
ListM1 = PM1.generate_samples(number) # in Msun
ListtM1=ListM1.tolist()
```

The mass of the secondary ZAMS star M_2 is determined by the distribution of initial mass ratio q_i

$$\xi(q_i) \propto q_i^{-0.1} ,$$
$$q_i = \frac{M_2}{M_1} \in [0.1, 1] \quad \text{and} \quad M_2 \geq 10 M_\odot .$$

based on

[20]:
```
# Generate mass ratio q:  a power law distribution, with power law index = -0.1

from scipy.stats import powerlaw

Listq=powerlaw.rvs(a=0.9,loc=0.1,scale=0.9,size=number,random_state=None)
Listtq=Listq.tolist()

#Thus the distribution of secondary ZAMS mass M2 is
ListM2=ListM1*Listq # in Msun
ListtM2=ListM2.tolist()
```

3 Initial Orbital Properties

The orbital properties considered are the period of the binaries P_i and their eccentricities e_i that are drawn from the following probability distributions

$$\xi(\mathcal{P}_i) \propto \mathcal{P}_i^{-0.55} , \quad \mathcal{P}_i \equiv \log_{10} \frac{P_i}{\text{day}} \in [0.15, 5.5] ,$$
$$\xi(e_i) \propto e_i^{-0.42} , \quad e_i \in [0, 1] .$$

[19]:
```
# Generate period: make a power law distribution, with power law index = -0.55
ListP1=powerlaw.rvs(a=0.45,loc=0.15,scale=5.35,size=number,random_state=None) #␣
 ↪In Log(P/days)
ListP=(10**ListP1)*DAY # In second
ListtP=ListP.tolist()

#Generate eccentricity: make a power law distribution, with power law index = -0.
 ↪42
Liste=powerlaw.rvs(a=0.58,loc=0,scale=1,size=number,random_state=None)
Listte=Liste.tolist()
```

[22]:
```
# Calculate semi major axis using period
Listp=(((ListP/(2*np.pi))**2 * G * (ListM1*2e30) * (Listq+1) *␣
 ↪(1-Liste**2)**3)**(1/3))/(696340* 10**3) # Semi letus rectum in Rsun
Listsemi=Listp/(1-Liste**2) # in Rsun
Listtsemi=Listsemi.tolist()
```

4 Black Hole Spin Distribution

We can choose from zero spin distribution (all stellar has zero spin), maximal spin distribution (all stellar has dimensionless spin 1), uniform spin distribution (spin distributed uniformly from 0 to 1). In our simulation, we choose uniform spin distribution.

[23]:
```
#Create three types of spin list

from scipy.stats import uniform

ListZeroSpin=np.zeros(number)
ListMaximalSpin=np.ones(number)
ListUniformSpin=uniform.rvs(loc=0,scale=1,size=number,random_state=None)
ListtZeroSpin=ListZeroSpin.tolist()
ListtMaximalSpin=ListMaximalSpin.tolist()
ListtUniformSpin=ListUniformSpin.tolist()
```

5 Metallicity

We draw a sample of metallicities following the distribution reported in Figure 5 of Lagarde et al., 2021. Defining $\mathcal{Z} = [\mathrm{Fe/H}]$ as the metallicity, they obtain the distribution in their Figure 5 that we are able to fit with the sum of a Gaussian and a power-law function

$$\begin{aligned}
&\xi(\mathcal{Z}) = \xi_1(\mathcal{Z}) + \xi_2(\mathcal{Z})\,, \text{ with } \mathcal{Z} \in [0, 0.5] \text{ and} \\
&\xi_1(\mathcal{Z}) = 0.03 \times \mathcal{Z}^{0.45} + 0.004\,, \\
&\xi_2(\mathcal{Z}) = 0.11 \times \exp\left[\frac{(\mathcal{Z} - 0.61)^2}{2 \times 0.008^2}\right]\,.
\end{aligned}$$

We have converted the Iron abundance metallicities in units of solar metallicities using $Z = Z\_\odot \times 10^{\mathcal{Z}}$, with $Z_\odot = 0.02$.

Lagarde et al., 2021: https://doi.org/10.1051/0004-6361/202039982

[25]:
```
# define a gaussian funtion with parameters A,X_mean, sigma
def gaus(X,A,X_mean,sigma):
    return A*np.exp(-(X-X_mean)**2/(2*sigma**2))

# define a powerlaw distribution with parameters A, a, cut, offset
def power(X,A,a,cut,offset):
    flag=(X<cut).astype(int)
    return flag*A*X**a+offset

# define the distribution combine the Gaussian and power-law function using the␣
 ↪definition above
def dist(X):
    return gaus(X,0.11239582,0.61297987,0.10741941)+power(X,0.03060204,0.
 ↪45446939,0.5,0.00395605)
```

```
# Distribution of metallicity
PZ=SamplingProbability(dist, [0,1] )

#List of Metallicity
ListZ1=PZ.generate_samples(number)
ListZ2=ListZ1*1.339912723097303-0.8755311276207431
ListZ=0.02*10**ListZ2
ListtZ=ListZ.tolist()
```

6 Other Setting For Simulation

[26]:
```
# Other list in str type

# Supernova type: choose from rapid, delayed, rapid_gauNS, delayed_gauNS,␣
 ↪compact, directcollapse, deathmatrix
Listsn=['rapid'] * number
# Choose initial stellar age: choose from zams,tams, shb, cheb, tcheb, sheb
Listtstart=['zams'] * number
# Choose end time of the simulation: choose from end or broken or specific time
Listend=['end'] * number
# Time shcedule for output: choose from List, Interval, all, end, events,␣
 ↪eventsrlo or input specific time
Listtout=['end'] * number
```

7 Construct the Initial Dataset

[28]:
```
#Remember to choose a spin distribution here
import pandas as pd
dataframe1=pd.DataFrame({'M1': ListtM1,'Z1':ListtZ,'Omega1':
 ↪ListtUniformSpin,"sn1":Listsn,'tstart1':Listtstart,'M2':ListtM2,'Z2':
 ↪ListtZ,'Omega2':ListtUniformSpin,'sn2':Listsn,'tstart2':Listtstart,'a':
 ↪Listtsemi,'e':Listte,'tend':Listend,'dtout':Listtout})
```

[29]:
```
dataframe=dataframe1[dataframe1['M2']>=10]
dataframe
```

[29]:

	M1	Z1	Omega1	sn1	tstart1	M2	Z2	\
0	12.811577	0.015407	0.910202	rapid	zams	10.131569	0.015407	
1	17.052604	0.003972	0.700718	rapid	zams	16.397239	0.003972	
4	23.459817	0.008974	0.280005	rapid	zams	18.799192	0.008974	
6	32.351466	0.019043	0.606185	rapid	zams	28.270256	0.019043	
7	31.635051	0.023284	0.928034	rapid	zams	13.392434	0.023284	
...	...	...	...	...	...	...	...	
59999993	21.775334	0.004185	0.889787	rapid	zams	10.345964	0.004185	
59999994	58.551411	0.017409	0.064217	rapid	zams	58.337987	0.017409	

```
59999996  27.193589  0.021608  0.680499  rapid    zams  19.646346  0.021608
59999997  20.985800  0.011587  0.247865  rapid    zams  11.111511  0.011587
59999999  34.106825  0.034348  0.501975  rapid    zams  15.136569  0.034348
```

```
            Omega2    sn2 tstart2             a         e tend dtout
0         0.910202  rapid    zams     15.941487  0.375583  end   end
1         0.700718  rapid    zams     27.928627  0.002581  end   end
4         0.280005  rapid    zams     23.218480  0.276763  end   end
6         0.606185  rapid    zams    616.847289  0.259445  end   end
7         0.928034  rapid    zams    219.268441  0.802589  end   end
...            ...    ...     ...           ...       ...  ...   ...
59999993  0.889787  rapid    zams  10908.149707  0.924916  end   end
59999994  0.064217  rapid    zams   4186.689297  0.640620  end   end
59999996  0.680499  rapid    zams   1227.371023  0.381645  end   end
59999997  0.247865  rapid    zams     16.860073  0.874193  end   end
59999999  0.501975  rapid    zams    224.474532  0.345385  end   end

[27174274 rows x 14 columns]
```

8 Save the Initial Data for One Kernal

[30]:
```
dataframe.to_csv('listBin_v1.dat',sep=' ',header=None,index=None)
```