


```
In [ ]: !pip install numpy==1.15.0
```

```
In [ ]: import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

```
In [ ]: ### Running in Google Colab? You'll want to uncomment and run these cell once each

"""
!wget https://raw.githubusercontent.com/psheehan/CIERA-HS-Program/master/Projects/B
!wget https://raw.githubusercontent.com/psheehan/CIERA-HS-Program/master/Projects/B
!wget https://raw.githubusercontent.com/psheehan/CIERA-HS-Program/master/Projects/B
!wget https://raw.githubusercontent.com/psheehan/CIERA-HS-Program/master/Projects/B
"""
```

 No description has been provided for this image

0. Binary Stars!

Binary stars are very common in the universe. As a graduate student my work focus on binary stars that evolve to become merging binary black holes!

This project is intended to show you one way astronomers use observations of binary black hole mergers to inform our understanding of the universe.

0a. Watch this "Crash Course" video on binary stars.

<https://www.youtube.com/watch?v=pIFiCLhJmig>

0b. Watch this video too. It goes more into detail about how binary stars can interact with one another.

At time=3:46 in the video talks about the complicated evolution of binary stars.

This is type of evolution is what allows for the formation of binary black hole systems!

<https://www.youtube.com/watch?v=AnE9EYgXxpU>

~~ Other cool videos about black holes to watch on your own:

2017: Graduate student Katie Bouman describing how her team will take a picture of a super massive black hole <https://www.youtube.com/watch?v=BlvezCVcsYs>

2019: Vox video showing <https://www.youtube.com/watch?v=pAoEHR4aW8I>

In this notebook you will be using simulation data from the code COSMIC.

This code is used to simulate binary evolution and was developed by graduate students, post-docs, and faculty in CIERA.

It builds on previous work to model complicated binary evolution very quickly.

Ideally we would model the complicated stellar interactions using fluid simulations like this <https://aasnova.org/2016/02/29/featured-image-orbiting-stars-share-an-envelope/> but we don't fully understand the physics involved and those simulations take a long time to finish.

The data I am giving you starts with a large initial population of stars with initial conditions that are supposed to replicate binary systems in the universe. COSMIC evolves those binaries through time.

0c. When binary stars were formed in the universe what do you think they looked like? Go on Slack and answer the first 2 anonymous polls. You aren't supposed to know the answer. This is just to get your brain thinking about binary stars!

1. Histograms

We are going to be using histograms a lot in this exercise to visualize our binary populations. 1a is supposed to help you become comfortable with them.

A histogram is a helpful way to bin a list of numbers.

1a. use `np.random.randint` to make a random list of numbers.

low = lower bound for random numbers

high = upper bound for random numbers

size = how many numbers you want, start with something >20

```
In [ ]: # make a random list of integers between 2 numbers
data = np.random.randint(low=, high=, size=)
print(data)
```

1b. Use plt.hist to bin your random numbers

data = list of random numbers

bins = number of bins

```
In [ ]: plt.figure()  
# feed in the array named data, place the values into 5 bins  
h = plt.hist(data, bins=5)
```

1c. Change the number of bins to a large number to see what happens to the data.

```
In [ ]: # change the number of bins here  
plt.figure()  
h = plt.hist(data, bins=)
```

Okay, lets finally read in the COSMIC data.

```
In [ ]: # There are three different sections to the data  
# You will learn what they all mean separately!  
  
binaries_initCond = pd.read_hdf('data.h5', key='initCond')  
binaries_bcm = pd.read_hdf('data.h5', key='bcm')  
binaries_bpp = pd.read_hdf('data.h5', key='bpp')
```

binaries_initCond

Gives information about the initial conditions of the binaries. binaries_initCond will show you one possible answer to the questions on the polls.

You can look at all the available data by seeing what columns you can access:

```
In [ ]: binaries_initCond.columns
```

You can go to this website to see what some of these columns mean: https://cosmic-popsynth.github.io/docs/v3.2.0/output_info/index.html

You can Ctrl+F or Command+F to search for the column name

Play around with the data

1d) Make histograms using plt.hist and plots using plt.scatter to see the initial conditions of these binaries.

1e) Write a sentence or two about what you see.

For example, if you plot `plt.hist(np.log10(binaries_initCond['porb']), bins=10, color='gray')` you can write something like:

"This is the distribution for the initial periods of the binaries. Large periods mean they start far away from each other, small periods mean they start close together. Most of the binaries start with small or medium valued orbital periods and fewer start with large orbital periods"

check if this is true!

**Some other interesting columns are 'ecc', 'mass0_1, and
binaries_initCond['mass0_2']/binaries_initCond['mass0_1']**

Bonus

Make your own histogram function! (It's a little more ticky than you might think!) There are so many functions that are already made for you but there is value in building them yourself! This is the best way to really understand what the function is doing

```
In [ ]: def plot_hist(data, bins ):
        return plt, bin_vals

data = np.log10(binaries_initCond['porb'])
bins = 20
plt, bin_vals = plot_hist(data, bins )
plt.xlabel('initial orbital period [log(days)]')
```

```
In [ ]: h = plt.hist()
plt.xlabel(, size=18)
plt.ylabel(, size=18)
```

```
In [ ]: h = plt.hist()
plt.xlabel(, size=18)
plt.ylabel(, size=18)
```

```
In [ ]: h = plt.hist()
plt.xlabel(, size=18)
plt.ylabel(, size=18)
```

```
In [ ]: # use this cell to check the answer to Poll #2
# to check it you need to plot the mass ratio, q
# q = m2/m1 (smaller mass/ larger mass)
# values close to 1 mean the masses are similar
# values close to 0 mean the masses are very different

h = plt.hist()
```

```
plt.xlabel(, size=18)
plt.ylabel(, size=18)
```

```
In [ ]: plt.scatter(, , alpha=0.5, s=0.1)
plt.xlabel(, size=18)
plt.ylabel(, size=18)
```

```
In [ ]: plt.scatter(, ,
                    alpha=, s=)
plt.xlabel(, size=18)
plt.ylabel(, size=18)
```

2 binaries_bcm

binaries_bcm has information about the last timestep in the data, tphys=13700.0.

It can tell you things like: There is a black hole with a white dwarf. Their orbital period is 5 solar radii and their eccentricity is 0.2

Let's look at the columns again

```
In [ ]: binaries_bcm.columns
```

```
In [ ]: binaries_bcm
```

```
In [ ]: # this is how you would look at all the values from 'RROL_2'
binaries_bcm['RROL_2'].values
```

Using **binaries_initCond** and **binaries_bcm** we can see the initial and final snapshot of each binary.

2a) Using binaries_initCond find the total mass in stellar objects in the initial conditions

for binaries_initCond you will want to use the columns mass0_1 and mass0_2

you can use np.sum() to sum up all the values

```
In [ ]: mass1_initial_total =
mass2_initial_total =

total_initial_mass =
print(total_initial_mass)
```

2b) Go on Slack and fill out the anonymous Poll #3. It asks how much initial mass you found. What are the units?

2c) Using `binaries_bcm` find the total mass in stellar object in the final conditions

for `binaries_bcm` you will want to use the columns `mass_1` and `mass_2`

```
In [ ]: mass1_final_total =  
        mass2_final_total =  
  
        total_final_mass =  
        print(total_final_mass)
```

2d. Find the ratio of final mass in all binaries to the initial mass.

This gives you the fraction of mass that is remaining. It should be less than one.

```
In [ ]: fraction = total_final_mass/total_initial_mass
```

2e Fill out anonymous Poll #4 requesting ideas about how the mass was ejected.

Lets look at specific rows in `binaries_bcm`

How do we access the same binary in `binaries_initCond` and `binaries_bcm`?

Each binary has its own unique number. In this data frame the unique number is its index value (the first column called `bin_num`). Run this cell to see

```
In [ ]: binaries_bcm
```

You can access the unique index numbers by doing the following

```
In [ ]: # binaries_bcm.index gives you the numbers, adding .values at the end puts it in to  
        # binaries_bcm.index.values  
  
        binary_index_vals = binaries_bcm.index.values  
        binary_index_vals
```

```
In [ ]: # you can access the elements in the array by doing:  
  
        print('0th element', binary_index_vals[0])  
        print('1st element', binary_index_vals[1])
```

If you have the row's index number (`binary_index_vals = binaries_bcm.index.values`) you use `.loc` to access the data for that index

```
In [ ]: print(binaries_bcm.loc[binary_index_vals[0]])

# remember binary_index_vals[0] will give you the unique binary values
```

Lets look at the final and initial values for one binary

```
In [ ]: print('Looking at initial conditions:')
print(binaries_initCond.loc[binary_index_vals[2]], '\n')

print('Looking at final conditions:')
print(binaries_bcm.loc[binary_index_vals[2]])

# you might get some weird output for the binaries_initCond. This is because I am u
# and the developed probably fixed that in the newer version
```

IMPORTANT: look before moving on

So far you have looked at entire columns in the data but what if you only want one row in one column?

.iloc and .loc are used for that **BUT** .iloc and .loc are very different.

.iloc doesn't care about the index (in this case the unique bin_num) of the row. It only cares about the position of the row in the data, similar to how you access a normal array. Use this when you want to call data as if it were in an array

.loc is used with the row index (in this case the unique bin_num).

make sure you understand this before moving on!

There are different ways to access specific rows in your columns. These are the ways it makes the most sense to me:

```
In [ ]: # here I look at the column named rad_1
print('Entire column:')
print(binaries_bcm['rad_1'])

# putting the unique index/bin_num values in an array
binary_index_vals = binaries_bcm.index.values

# here I look at the column rad_1 but only the first row.
# to access the first row with loc we use the row's index value: index value = bina
# remember, since we are using the index values we have to use loc, not iloc
print(' -----')
print('Just first row with .loc')
index_value = binary_index_vals[0]
print('index value = ', index_value)
print(binaries_bcm['rad_1'].loc[index_value])
```

```
# to access the first row with iloc we can access it like a numpy array
print(' -----')
print('Just first row with .iloc')
i_value = 0
print(binaries_bcm['rad_1'].iloc[i_value])
```

Another important note:

Think carefully when you want to use .iloc or .loc.

If you are just looking in one data frame then .iloc is probably fine.

If you want to access the same binary in the dataframe binaries_bcm and binaries_initCond it is better to use .loc. You can use .iloc BUT are you sure that .iloc[0] will give you the same binary every time? It is much more safe to use .loc because you know that the index values are unique to each binary.

3. Mass in black holes

The point of this section is to get more comfortable with the accessing the data.

3a) Use binaries_bcm to find the total mass in the last snapshot that is only in black holes.

Tip: You can do this using a for loop but it can be done MUCH faster if you use something like np.where() Look at this stack overflow link on np.where

<https://stackoverflow.com/questions/34667282/numpy-where-detailed-step-by-step-explanation-examples>

You can find which objects are black holes using the "kstar_1" and "kstar_2" columns. Here is what the numbers represent.

https://cosmic-popsynth.github.io/docs/v3.2.0/output_info/index.html

```
In [ ]: # here is a mini tutorial if you are stuck using np.where

a = np.arange(0,10)
print('a =', a, '\n')

print('I want to find the indicies where this condition is met a<5.')
print('a<5 =', a<5, '\n')

print('Using np.where like this will give you the indicies where the condition is met')
where_true = np.where(a<5)[0]
print('np.where(a<5)[0] = ', where_true, '\n')
```



```
print('Now I can go back to my array "a" and call those elements')
print(a[where_true])
```

In []:

In []:

In []:

3b) What is the fraction of BH mass at the end of simulation to the total mass at the end of the simulation?

In []:

3c) Go to Slack and answer anonymous Poll #5 asking what fraction you got.

In []:

In []:

4. A look at the bpp

We need to learn how to use binaries_bpp

binaries_bcm and binaries_iniCond have just one row for each binary.

binaries_bpp contains multiple rows for each binary. Every row is a different time and represents a key evolutionary change.

In []: `binaries_bpp`

Lets once again get the unique index values so that we can access different binaries.

In []: *# put the index values in a numpy array for easy access*
`binary_index_not_unique_values = binaries_bpp.index.values`
`binary_index_not_unique_values`

In []: *# this time we need to use np.unique since many rows have the same index*
`binary_index = np.unique(binaries_bpp.index.values)`
`binary_index`

In []: *# remember, since we are using the index values we have to use loc, NOT iloc*
`binaries_bpp.loc[binary_index[0]]`

4a) Choose any binary you want. Make a plot showing how the mass of the primary star "mass_1" changes over time "tphys". Make sure you label you plots! What are the units?

Tips:

You can access the data by doing:

```
binaries_bpp["column_name"].loc[binary_index_num].values
```

```
In [ ]: x =  
y =  
  
plt.plot(x, y)  
plt.xlabel(, size=18)  
plt.ylabel(, size=18)
```

4b) Make a plot showing how the mass of the primary star "mass_1" and secondary star "mass_2" changes over time "tphys". Add labels so you can tell them apart

Look through the binaries until you find a system like this:



No description has been provided for this image

One mass is increasing while the other decreases. From the videos you watched, can you describe what is happening?

4c) Go to Slack and fill out the anonymous free response poll (Poll #6) asking what binary interaction explains the changes in mass

```
In [ ]: i = 1  
x =  
y1 =  
y2 =  
  
plt.plot(x, y1, label=, linewidth=3)  
plt.plot(x, y2, label=, linewidth=3)  
plt.xlabel(, size=18)  
plt.ylabel(, size=18)  
plt.legend()
```

5. Merging Binary Black holes

Watch these two videos about LIGO/Gravitational waves.

<https://www.youtube.com/watch?v=B4XzLDM3Py8>

<https://www.youtube.com/watch?v=4GbWfNHtHRg>

Now lets do something more challenging!

The goal of this section is for you to find out which binary black hole systems will merge within a Hubble time. What causes these black holes to merge? Gravitational waves emitted by the black holes orbiting each other take away energy from the orbit. When this happens the orbit shrinks. We can calculate how long it will take for two black holes to merge if gravitational waves are the only thing causing the orbit to lose energy.

Hubble time = the age of the universe = 13.8 billion years

You will need to find all the binaries where in the last timestep both objects are black holes. You then need to calculate the merger time (t_{merger}) with the equations provided.

5a) Calculate which binary black hole systems will merge within the age of the universe. Use the direction/tips below

You need to create 3 arrays.

- array to store the binary index of the merged binaries
- array to store the mass of the most massive BH before merger
- array to store the smaller mass BH before merger.

A guide/tips:

1) You will need to access each binary similar to how you made the plots:

`binaries_bpp["column_name"].loc[binary_index_number].values`

2) You need to check that both objects are black holes **kstar_1 == 14** and **kstar_2 == 14** in the last timestep

To look at the last element in an array you use `array[-1]`

Here it looks like `binaries_bpp['column_name'].loc[binary_index_number].values[-1]`

or

`np.array(binaries_bpp['column_name'].loc[binary_index_number])[-1]`

3) You need to get the values of eccentricity, separation, M1, M1, and time in the last timestep. Put them in CGS units

<http://www.astro.wisc.edu/~dolan/constants.html>

4) Calculate t_merger

For circular orbits

$$t_{\text{merger,circular}} = \frac{a^4}{4\beta}$$

where $\beta = \frac{64}{5} \frac{G^3 m_1 m_2 (m_1 + m_2)}{c^5}$, a is separation, G is the gravitational constant, and c is the speed of light. It is unlikely that a binary will be perfectly circular.

For eccentricity < 0.5 use:

$$t_{\text{merger}} = \frac{c_0^4}{4\beta} e^{48/19}$$

where

$$c_0 = \frac{a(1 - e^2)}{e^{12/19} \left[1 + \frac{121}{304} e^2 \right]^{870/2299}}$$

for eccentricity > 0.5 use:

$$t_{\text{merger}} = \frac{768}{425} t_{\text{merger,circular}} (1 - e^2)^{7/2}$$

These equations are from a paper from 1964!

<https://ui.adsabs.harvard.edu/abs/1964PhRv..136.1224P/abstract>

*** Notes: ***

- Some eccentricities = -1. This is not physical and means the black holes are not orbiting each other anymore. Find a way to ignore these.
- You will have to keep track of the units. To do this you can change all simulation variables to cgs. The output of your t_merger calculations will be in seconds

5) Add your calculated t_merger to the time in the last timestep. Check if that sum is less than the age of the universe, 13.8 billion years. (remember that your t_merger is in seconds)

6) If the time is less than 13.8 billion years, congratulations! You found a merging black hole binary system!

7) When you find a merging system save its data! In one array save the index of the merging binary. In another array save the mass of the more massive black hole before the merger, this is called the primary BH. In another array save the mass of the lower mass BH, this one is called the secondary

```
In [ ]: # it is much easier and neater to write up equations in a function
# when you are trying to calculate something many times

# see how you can make a function here
def number_squared( x ):
    return x**2

# you can call functions inside other functions
def divide_by_2_after_square(x):
    squared = number_squared(x)
    return squared/2

number = 10
print(number_squared( number ))
print(divide_by_2_after_square( number ))

print('*****NOTE*****')
print('this is NOT how you should calculate (x**2) / 2')
print('this is just an example of how you may use functions for this exercise')
```

```
In [ ]: # Here is the function for beta to get you started, you just need to fill in G and
def calc_beta(m1,m2):
    G =
    c =
    return (64./5.) * ( (G**3 * m1 * m2 * (m1+m2) ) / c**5 )

def calc_t_merger_small_e():

    return

def calc_t_merger_big_e():

    return
```

```
In [ ]: Msun = 1.99e33
Rsun = 6.96e10
sec_in_year = 3.154e7

# this is here incase you need something to get your started. you don't have to use

merging_binaries=[]
primary_mass = []
secondary_mass = []

for in :

    if binaries_bpp['kstar_1'].loc[index].values[-1] == 14:
```

5b) Find the fraction of binaries from your initial population that resulted in BBH mergers.

- Sum up the total number of binary black hole mergers.
- Divide that number by the total number of initial binaries in the population.
 - Hint: Remember that each binary has a unique index.

In []:

5c) Go to Slack and fill out the anonymous poll 7 asking: what fraction of the initial binaries merged as BBHs.

6a. Use plt.hist to compare the initial conditions of the binaries that ended up merging to the initial conditions of the entire population

Tips:

- you should probably make two different histograms inside of plotted one on top of the other
- remember you can use the binary index and .loc to get specific binaries

```
In [ ]: plt.figure()
h = plt.hist(, range=())
plt.xlabel(, size=18)
plt.ylabel(, size=18)
plt.title('initial population', size=18)

plt.figure()
h = plt.hist(, range=())
plt.xlabel(, size=18)
plt.ylabel(, size=18)
plt.title('initial condition of bbh merger population', size=18)
```

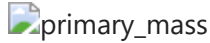
In []:

6b) Make a histogram of only the primary masses you saved (the more massive BH).

- remember, you were asked to save the index of the binaries that merged and the mass of the more massive black hole before the merger

```
In [ ]: h = plt.hist()
plt.xlabel(, size=18)
plt.ylabel(, size=18)
plt.title('primary mass before BBH merger', size=18)
```

6c) compare your plot to this one.



This plot shows the rate of BBH mergers per volume per year per primary mass. You can think of this plot as how likely you are to see a binary black hole merger with a specific primary mass. This plot was made using LIGO observations of real BBH mergers. The different colors correspond to different models. Each model has different assumptions, just to name a few: Model A assumed that BHs with masses below $5M_{\odot}$ do not exist. Model B assumed a different initial distribution of black holes.

Your histogram is similar to this plot because a histogram can tell you there are many more mergers with primary masses around $5M_{\odot}$. This is similar to saying, if you had to guess it is more likely you will see a BBH merger with a primary mass around $5M_{\odot}$.

This is not a perfect comparison but it is good enough! **What comments can you make about it?** Remember you are comparing simulation data to data informed by real events!

This figure comes from this paper:

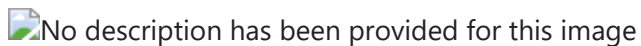
<https://dcc.ligo.org/public/0156/P1800324/009/O2RandP.pdf> you don't have to read it. it is just for your reference

6d) Make a histogram of the mass ratio, $q = \text{secondary_mass}/\text{primary_mass}$

If your numbers are in an array you can simply divide the arrays.

```
In [ ]: q =
```

6e) Compare your plot to this one



This plot shows the rate of BBH mergers per volume per year per **mass ratio**. This is the same as the first plot. Again, you can think of this plot as how likely you are to see a binary black hole merger with a specific **mass ratio**. The colors here correspond to the same models as above.

What comments can you make about this comparison? Do any of the models look similar to your histogram?

6f) Read this short article on the mass distribution of the resulting BH mass after a merger.

<https://aasnova.org/2019/07/10/exploring-a-black-hole-mass-conundrum/>

6g) Make a histogram of the resulting BH mass in each merger.

If your numbers are in an array you can simply add them. Adding arrays will add the numbers element by element

```
In [ ]: total_mass =
```

```
In [ ]: h = plt.hist()  
plt.xlabel(, size=18)  
plt.ylabel(, size=18)  
plt.title(, size=18)
```

6h) Compare your plot to this



No description has been provided for this image

This is the last plot from the article you read.

It comes from this scientific paper <https://arxiv.org/pdf/1901.03345.pdf>. You don't have to read this

In this plot the y-axis shows a similar idea to the last image. It shows model predictions for the distribution of the observed total mass of 10 mergers. Similar to the last image you can think of it as how likely it is to see a certain distribution of total final BH masses. The different colors are again different models. The black line is informed by the observed values from LIGO detections.

This is not a perfect comparison but it is good enough! What comments can you make about your total mass distribution and the figure? What range do you have on the x-axis and what range does the other plot have?

7. Do this same comparison with the different initial populations I have provided.

You do not have to redo all the poll questions.

This is meant for you to see how your results can change if you make different assumptions about the initial population.

Changing parameters like this is one way astronomers can tune their assumptions to better match observations. This helps us have a better understanding of the universe by ruling out assumptions that don't match observations.

data_diff_imf.h5 has a completely different initial population of the binaries. The initial orbital periods, eccentricities, and mass ratios, among others.

data_lowZ.h5 has a different metallicity. It is 100x smaller amount of metals

data_wkicks.h5 this one includes "kicks". When stars explode as super nova the explosion can "kick" the remaining black hole.

Which file gave a better comparison to the plots from scientific papers?

Answer the final poll (yay!), Poll #8.

In []: