1 Merger Trees

Here we describe the implementation of time delays in the evolution of a black hole along its merger history (merger tree). We do so for all black holes that result at present day ($z = 0$) with a mass greater or equal to $10^8 M_\odot$. We take data from the De Lucia catalogue of the Millennium Simulation.

- We find all the galaxyId and the corresponding lastProgenitorId for all the galaxies at the last snapnum possessing a black hole with $M \geq 10^8 M_\odot$ (see A.1 for the query bash script). All the galaxies with an identifier comprised between that of the root galaxyId and its lastProgenitorId belong to the merger tree of the root galaxy. Among these, we only consider those mergers, where the progenitor black holes have masses greater than zero. We obtain 190501 potential trees.

- Using the data from the previous query, we get the mergers in the merger trees for all the galaxies found at the previous step (see A.2 for details). The mergers are snapnum ordered, with the oldest mergers first.

- Empty files (trees with no mergers, given the conditions set in A.2) are eliminated (8631 empty trees). All those left are stored in a unique file with each tree separated from another by a -1 raw (see an example below). Therefore all the mergers comprised between two -1 raws refer to a single tree and are ordered from the most ancient to the most recent. The total amount of mergers is 587607

```
galaxyId,snapnum,P1_Id,P2_Id,z,...
693,27,694,746,3.060419,...
993,27,994,1033,3.060419,...
30,33,31,327,1.9126327,...
28,35,29,415,1.6302707,...
25,38,26,683,1.2758462,...
24,39,25,836,1.173417,...
24,39,25,880,1.173417,...
24,39,836,880,1.173417,...
22,41,23,980,0.98870814,...
20,43,21,1215,0.8276991,...
19,44,20,1335,0.75503564,...
14,49,15,1892,0.45657724,...
13,50,14,2013,0.40789944,...
7,56,8,2316,0.17489761,...
4,59,5,2594,0.08928783,...
1,62,2,2881,0.019932542,...
1,62,2,2941,0.019932542,...
1,62,2881,2941,0.019932542,...
-1,-1,-1,-1,-1,...
15478,35,15479,15530,1.6302707,...
15477,36,15478,15573,1.5036365,...
15468,45,15469,15661,0.6871088,...
-1,-1,-1,-1,-1,...
...
```

- Multiple trees, however, contain multiple mergers, i.e. more than 2 galaxies (therefore more than two black holes) merge at the same snapnum to give the same resulting galaxy. That means that in a given tree there could be more lines containing the same galaxyId and accounting for multiple mergers (the number of lines corresponds to the number of ways one can choose a couple of black holes from the set of merging black holes, i.e. 3

lines if 3 black holes merge, 6 lines if 4 black holes merge and so on) They are resolved so that there is only one merger for each galaxyId present in a tree, before implementing time delays. The details of the treatment are described in the next section.

1.1 Multiplets treatment in raw data

Multiple mergers at same snapnum are first ordered so that the first three account for the merger of the three most massive black holes in the set. Say $m_1 > m_2 > m_3$ are the three most massive black holes in a multiple merger, then the first three lines will contain the mergers

1. $m_1 + m_2$
2. $m_1 + m_3$
3. $m_2 + m_3$

All the subsequent lines, (in case more than 3 black holes merge at the same time) are eliminated. Thus we remain with a total number of 28390 times when the same galaxyId is listed for three times. We refer to these as triplets at same snapnum.
At this point the most massive combination (the first raw as the triplet are mass ordered) is taken to be the inner binary, whereas m_3 is treated as the intruder. The triplets are then processed with the Bonetti program, which in 20% of cases resolves the triplets and gives back a binary merger, either as a prompt merger of two of the black holes (the remnant forms a binary with the third one) or as an ejection of one of the black holes (the remaining two form a binary). Non resolved triplets are manually discharged leaving the most massive merger only (the first line).

N.B. I'm not sure about this procedure as there really is no inner binary and an intruder, as all the three black holes merge at the same time. So maybe it's better to keep the most massive merger from the start...

1.2 Triplets

The starting data contain trees with a single merger for each galaxyId and the total amount of mergers is 530827.
At these point, each tree is analyzed separately (it's easily accessible as it is between two -1 raws).

```
i=0
while i<len(galaxyId):
  j=i
  while (galaxyId[j]!=-1):
    j=j+1
    if (j>=(len(galaxyId))):
      break
  tree_index=j
  # Analyze individual trees
  for k in range(i, tree_index):
    ...

  i=tree_index+1
  if (i>=len(galaxyId)):
    break
```

We create 4 vectors (*type_P*1, *type_P*2, *P*1*_marker* and *P*2*_marker*) to store data which will allow us to analyze the tree. They are vectors long as the total number of lines of the data file, so that data at the same index correspond to information on the merger at that index.

*type_P*1 and *type_P*2 are vectors that store information on the two progenitors, P1 and P2, both initialized with 0 values

- If the value is 0 then the progenitor is a single black hole
- If the value is 2 then the progenitor is a binary (it means that it hasn't successfully merged before the subsequent galaxy merger)

Therefore we can have 4 different combinations depending on the type of the progenitors

1. *type_P*1 = 0 and *type_P*2 = 0: the progenitors are both single black holes, we have a **binary**!
2. *type_P*1 = 2 and *type_P*2 = 0: the first progenitor is a binary (it will be the inner binary in the Bonetti program) and the second is a single balck hole (the intruder). We have a **triplet**!
3. *type_P*1 = 0 and *type_P*2 = 2: the first progenitor is a single black hole (the intruder) and the the second progenitor is a binary (the inner binary). We have a **triplet**!
4. *type_P*1 = 2 and *type_P*2 = 2: we have a **quadruplet**! In this case the least massive black hole is manually eliminated. We are left with a triplet. If the least massive black hole comes from the P1 binary, then the P2 binary is the inner binary and the left over black hole from the P1 binary becomes the intruder, and vice versa in other case.

Except for the first case, we have to implement the triplet interaction in all the other cases. *P*1*_marker* and *P*2*_marker* store the index on where to find the information on the progenitors in case these are NOT single black holes.

Follows a sketch of the algorithm.

1.2.1 Binary mergers

We sketch the first type merger, where both progenitors are single black holes (*type_P*1 = 0 and *type_P*2 = 0):

- So suppose we have a binary merger at $z_{\rm i}$ (both progenitors are single black holes). This binary merger is stored at index $k_{\rm i}$ of the data.
- We calculate the time delay, $T_{\rm delay}$ necessary for the two central black holes to merge (see notes on time delay for the formulae employed)
- Then we search the tree to find its descendant. This is pretty easy as galaxyId in a tree are ordered for a depth-first search, that means that if the difference of two galaxies Id equals the difference in snapnums then the galaxy at higher snapnum[1] descends from the galaxy at lower snapnum

[1]Snapnums are ordered from 0 to 63, where $snapnum = 63$ corresponds to present day.

```
# Find descendant function
def find_descendant(k, tree_index):
  P1=0
  P2=0
  descendant_index=-1
  for l in range(k+1, tree_index):
    if ((snapnum[l]-snapnum[k])==(1+galaxyId[k]-P1_galaxyId[l])):
      P1=1
      z1=redshift[l]
      descendant_index=l
      break
    if ((snapnum[l]-snapnum[k])==(1+galaxyId[k]-P2_galaxyId[l])):
      P2==1
      z1=redshift[l]
      descendant_index=l
      break
  info_descendant=np.array([descendant_index,P1,P2])

  return info_descendant
```

The function takes as input the index at which the galaxy is stored, k, and the index, *tree_index* at which the next -1 raw is. It gives back the index at which the first descendant is and specifies whether the descendant is the first, P1, or the second progenitor, P2.

- At this point we calculate the time elapsed between the redshift, $z2$, at which this merger happens and the redshift, $z1$, at which the descendant merger will occur (for the calculation see the paragraph 1.3.1). We call this time $T_{\text{next\_merger}}$.

- If $T_{\text{delay}} > T_{\text{next\_merger}}$, then at the subsequent merger either one of the progenitors will be a binary and we already have all the information to update the four vectors above, *type_P1*, *type_P2*, *P1_marker* and *P2_marker*.

```
if(info_descendant[0]==-1):
            break
if (info_descendant[0]!=-1 and info_descendant[1]==1 and info_descendant
    [2]==0):
            # P1 of descendant is a binary!
            type_P1[info_descendant[0]]=2
            P1_marker[info_descendant[0]]=k
if (info_descendant[0]!=-1 and info_descendant[1]==0 and info_descendant
    [2]==1):
            # P2 of descendant is a binary
            type_P2[info_descendant[0]]=2
            P2_marker[info_descendant[0]]=k
```

- Moreover, at the subsequent merger the binary that hasn't merged has updated masses thanks to the implementation of the radio mode (see 1.4)

```
if (time_delay>=time_next_merger):
        P1_BH_mass[k]=P1_BH_mass[k]+radio_mode.radio(time_between_mergers,k
    ,1,0)
        P2_BH_mass[k]=P2_BH_mass[k]+radio_mode.radio(time_between_mergers,k
    ,0,1)
```

Note that the progenitor masses are updated directly at the index k of the first merger, since $P1\_marker$ and $P2\_marker$ vectors store the index of the previous merger.

- On the other hand, if the binary successfully merge before the next merger, then the initial values of $type\_P1$ and $type\_P2$ are good (they already represent single black holes), and $P1\_marker$ and $P2\_marker$ will not be needed.

1.2.2 Triplet mergers

In case one of the two progenitors is a binary, we have to deal with a triplet interaction. This happens when either $type\_P1$ or $type\_P2$ equals 2. Let's analyze one case. The other is analogous

- We first recover the information on the progenitor that is still a binary, and resolve for the inner binary and the intruder information, that have to be passed to Bonetti's program

```
if (type_P1[k]==0 and type_P2[k]==2): # P1 is the intruder, P2 is a binary
    # Triplet

    # Binary from P1 and single BH (intruder) from P2
    # Find m_1, q_in and q_out to pass to triplet_function
    P1_index=-1
    P2_index=P2_marker[k]
    if(P1_BH_mass[P2_index]>P2_BH_mass[P2_index]):
      m_1=P1_BH_mass[P2_index]
      m_2=P2_BH_mass[P2_index]
    else:
      m_1=P2_BH_mass[P2_index]
      m_2=P1_BH_mass[P2_index]
    q_in=m_2/m_1
    m_3=P2_BH_mass[k]
    q_out=m_3/(m_1+m_2)
```

- Bonetti's program (see next paragraph 1.2.3) takes as input (m_1, q_in, q_out) and returns a number in $[1, 7]$. Each value corresponds to a different situation

  1. If $triplet\_output = 1$ then a prompt merger between m_1 and m_2 occurred. Then we assume that a binary forms between the merged $m\_1 + m\_2$ and m_3
  2. If $triplet\_output = 2$ then m_3 has been ejected and we assume that m_1 and m_2 may undergo a delayed merger
  3. If $triplet\_output = 3$ then a prompt merger between m_1 and m_3 occurred. Then we assume that a binary forms between the merged $m\_1 + m\_3$ and m_2
  4. If $triplet\_output = 4$ then m_2 has been ejected and we assume that m_1 and m_3 may undergo a delayed merger
  5. If $triplet\_output = 5$ then a prompt merger between m_2 and m_3 occurred. Then we assume that a binary forms between the merged $m\_2 + m\_3$ and m_1
  6. If $triplet\_output = 6$ then m_1 has been ejected and we assume that m_2 and m_3 may undergo a delayed merger
  7. If $triplet\_output = 7$ then triplet interaction hasn't been successful (see 1.2.5).

  The output is passed to an *output_analyzer* function that implemets the next steps for each case.

```
# Launch triplet interaction
        triplet_output=bonetti.triplet_function(m_1,q_in,q_out)
        output=triplets.output_analyzer(triplet_output,k,P1_index,P2_index,
    tree_index,time_to_sink)
```

- In all cases first the sinking time is calculated, i.e. the dynamical friction time it takes for the binary on one hand, and the intruder on the other, to sink to the center of the newly formed galaxy

```
time_to_sink=delay.dynamical_friction(BH_mass[P2_index],P1_BH_mass[k],\
P_stellar_mass,P_bulge_mass)
```

- At this point we have three possibilities. Cases 1), 3) and 6) are similar and the steps are the following
  - We assume that the prompt merger occurs after the binary and the intruder have reached the influence distance (dynamical friction time)
  - The new binary is then assigned a delay time from the hardening phase on (plus the sinking time)
  - As in the binary merger case above, we proceed searching for the next descendant and analyzing whether the merger will occur before the next galactic merger
- In cases 2), 4) and 6) after the *time_to_sink* has passed, one black hole is ejected and again we remain with a binary. In this case the delay time is randomly selected between 3×10^8 and the time to next merger (plus the sinking time)

```
time_between_mergers=lookback.time_between_mergers(z1,redshift[k])
time_to_merge=time_to_sink+random.uniform(start_time,time_between_mergers)
```

At this point the comparison between the delay time and the time to the next merger proceeds as for the binary system explained above.

1.2.3 Bonetti's program

```
def triplet_function(m_1,q_in,q_out):
  M_sun=M_sun=1.99*10**(30)
  input_data=([m_1/M_sun,q_out,q_in])
  input_data=np.log10(input_data)

  # Pass input data to the interpolation function for all 6 possibile cases
  prompt_prob12=trilinear_interp(input_data, m1, qout, qin, prompt_merger_frac12
    )
  prompt_prob13=trilinear_interp(input_data, m1, qout, qin, prompt_merger_frac13
    )
  prompt_prob23=trilinear_interp(input_data, m1, qout, qin, prompt_merger_frac23
    )
  delayed_prob12=trilinear_interp(input_data, m1, qout, qin,
    delayed_merger_frac12)
  delayed_prob13=trilinear_interp(input_data, m1, qout, qin,
    delayed_merger_frac13)
```

```
    delayed_prob23=trilinear_interp(input_data, m1, qout, qin,
      delayed_merger_frac23)

  # Create vector of outcome probabilities
    P12=prompt_prob12/100
    D12=P12+delayed_prob12/100
    P13=D12+prompt_prob13/100
    D13=P13+delayed_prob13/100
    P23=D13+prompt_prob23/100
    D23=P23+delayed_prob23/100
    no_interaction=1.0
    probability_vector=np.array([P12,D12,P13,D13,P23,D23,M0])


    # Extract a random number from 0.0 to 1.0
    random_num=np.random.random(1)

    if (random_num[0]<=probability_vector[0]):
      j=1 # prompt merger between m_1 and m_2
    if (random_num[0]>probability_vector[0] and random_num[0]<=probability_vector
      [1]):
      j=2 # delayed merger between m_1 and m_2 -> m_3 ejected
    if (random_num[0]>probability_vector[1] and random_num[0]<=probability_vector
      [2]):
      j=3  # prompt merger between m_1 and m_3
    if (random_num[0]>probability_vector[2] and random_num[0]<=probability_vector
      [3]):
      j=4  # delayed merger between m_1 and m_3 -> m_2 ejected
    if (random_num[0]>probability_vector[3] and random_num[0]<=probability_vector
      [4]):
      j=5  # prompt merger between m_2 and m_3
    if (random_num[0]>probability_vector[4] and random_num[0]<=probability_vector
      [5]):
      j=6  # delayed merger between m_2 and m_3 -> m_1 ejected
    if (random_num[0]>probability_vector[5] and random_num[0]<=probability_vector
      [6]):
      j=7 # still a triplet!

    return j
```

1.2.4 Quadruplets

Quadruplets happen when both the progenitors are binaries that didn't menage to merge in time, ($type\_P1 = 2$ and $type\_P2 = 2$). We have four black holes forming two binaries. We have to find out the least massive one to eject (which could be $mass1$, $mass2$, $mass3$ or $mass4$) and then launch the triplet interaction as explained in the previous section.

```
P1_index=P1_marker[k]
P2_index=P2_marker[k]
# sort out the least massive BH to eject
# from P1
mass1=P1_BH_mass[P1_index]
mass2=P2_BH_mass[P1_index]
# or from P2
mass3=P1_BH_mass[P2_index]
mass4=P2_BH_mass[P2_index]
```

1.2.5 Resolve unresolved triplets

In case of insuccess of the triplet interaction (the most probable scenario) we manually the two most massive black holes only and eject the third one. For what concerns the delay time, we do as follows

- Suppose a binary forms at $T_{\rm i}$ with a delay time of $T_{\rm delay}$
- $T_{\rm delay}$ is greater than the time to next merger, $T_{\rm next\_merger}$, which happens at time $T_{\rm j}$
- At $T_{\rm j}$ we have a triplet system
- After the Bonetti interaction the triplet still remains a triplet
- We manually discharge the least massive black hole and assign to the left over binary a delay time equal to the difference between $T_{\rm delay}$ previously calculated and $T_{\rm next\_merger}$
- Search for the next descendant and analyze whether the corresponding descendant will still be a binary or a single black hole

1.3 Record merger type

We have three vector to record whether each merger is a successful binary, a triplet with prompt merger or undergoes an ejection

```
binary_vector=np.zeros(len(galaxyId))
triplet_vector=np.zeros(len(galaxyId))
ejection_vector=np.zeros(len(galaxyId))
```

At the corresponding merger index, each vector is updates depending on the outcome of the binary. These then form the output data along with the redshift

```
for n in range(len(galaxyId)):
  if (galaxyId[n]!=-1):
    f.write(f'{P1_BH_mass[n]},{P2_BH_mass[n]},{redshift[n]},{binary_vector[n]},{
   triplet_vector[n]},{ejection_vector[n]}\n')
  else:
    f.write(f'-1,-1,-1,-1\n')
f.close()
```

Therefore, for each tree, we know whether each merger in the tree is a binary, a prompt triplet interaction or presents an ejection.

From our data we obtain that 85% of mergers are binaries. Of the remaining 15% triplets, 32% undergoes either a prompt merger (24%) or an ejection (8%).

1.3.1 Time to next merger

The time between mergers is calculated using

$$T_{\rm next\_merger} = t_{\rm H} \int_{z_1}^{z_2} dz' \frac{dz'}{(1+z')E(z')} \tag{1}$$

where $t_{\rm H}$ is the Hubble time[2]

$$E(z) = \sqrt{\Omega_{\rm M}(1+z)^3 + \Omega_\Lambda} \tag{2}$$

Here we have assumed a flat geometry universe[3] ($\Omega_{\rm k} = 0$)
We numerically integrate (1) using $N = 10^2$ points in the interval $[z_1, z_2]$.

```
# Useful constants
Gyr=3.15*10**16
TH0=13.4*Gyr
Omega_matter=0.25
Omega_lambda=0.75

# Lookback time integrand function
def integrand(z):
    f=1/((1+z)*np.sqrt(Omega_matter*(1+z)**3+Omega_lambda))
    return f

# Function to calculate the lookbacktime between z1 and z2
def time_between_mergers(z1,z2):
    N=10**2
    dz_vector=np.linspace(z1,z2,num=N)
    sum=0
    for i in range(len(dz_vector)):
        sum=sum+integrand(dz_vector[i])
    sum=(z2-z1)*sum/N
    time_next_merger=TH0*(sum)/Gyr
    return time_next_merger
```

1.4 Accretion during dynamical inactivity - radio mode

In the case the binary delay time is greater than the time to the next merger, a triplet forms.

- Suppose the binary, $m_1 - m_2$ forms at time $T_{\rm i}$ and it's characterized by a delay time given by $T_{\rm delay}$
- The time elapsed until the subsequent merger at $T_{\rm j}$ is $T_{\rm next\_merger} = T_{\rm j} - T_{\rm i}$
- The binary doesn't make it to coalesce in time, $T_{\rm delay} > T_{\rm next\_merger}$
- We have a triplet system at $T_{\rm j}$, with $m_1 - m_2$ constituting the internal binary and m_3 the intruder

To implement the interaction between m_1, m_2 and m_3, we use the mass for m_3 as given by the simulation, whereas the masses, m_1 and m_2, which are known thanks to the simulation at time $T_{\rm i}$ are updated to their new values by taking into account hot gas accretion (radio mode). We have that $m_{1,T_{\rm j}} = m_{1,T_{\rm i}} + \delta m_1$ and $m_{2,T_{\rm j}} = m_{2,T_{\rm i}} + \delta m_2$, where the radio mode accretion δm is given by

[2]The Hubble time is the inverse of present day Hubble constant and it represents the time elapsed from the origin of the universe to now. In the simulation $H_0 = 73 {\rm kms}^{-1}{\rm Mpc}^{-1}$ and therefore $t_{\rm H} = 13.4{\rm Gyr}$

[3]Otherwise, the complete function would have been

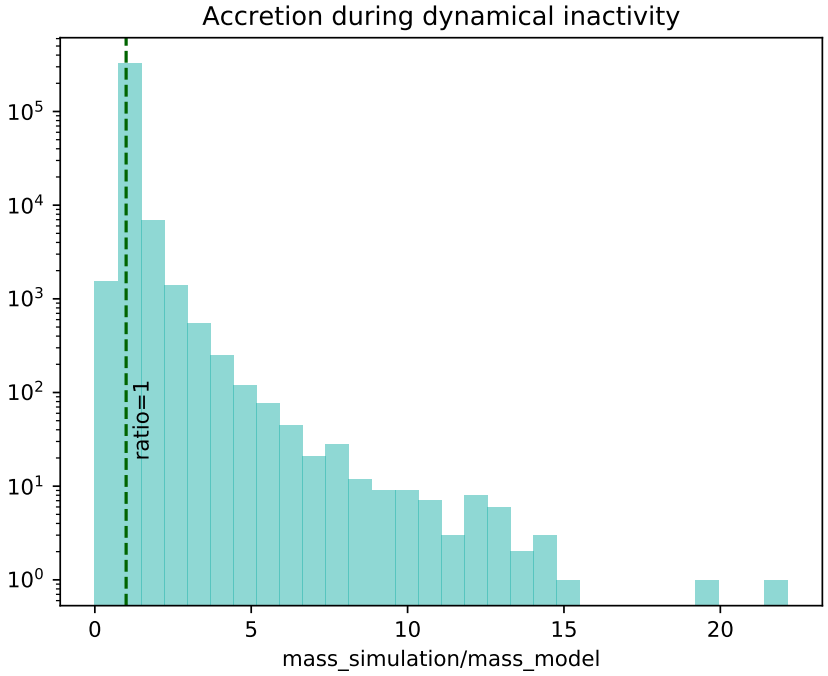
$$E(z) = \sqrt{\Omega_{\rm M}(1+z)^3 + \Omega_{\rm k}(1+z)^2 + \Omega_\Lambda}$$

$$\delta m_{\rm radio\_mode} = \kappa \left(\frac{f_{\rm hot}}{0.1}\right) \left(\frac{V_{\rm vir}}{200{\rm kms}^{-1}}\right)^3 \left(\frac{m}{10^8 h^{-1} M_\odot}\right) \left(\frac{T}{1{\rm yr}}\right) M_\odot \tag{3}$$

where

- κ sets the efficiency of hot gas accretion; in Guo2010 $\kappa = 1.5\times10^{-5}$ and it is $\kappa = 7.5\times10^{-6}$ in De Lucia (see [Guo 2010])
- $f_{\rm hot}$ is the ratio of hot gas mass, $M_{\rm hot}$, to the subhalo DM mass, $M_{\rm DM}$. There is a relation between the mass of a galaxy, M_* and the dark matter mass, $M_{\rm DM}$, given by (see [Arxiv1709.06501])
$$\left(\frac{M_{\rm DM}}{10^{13} M_\odot}\right) = 0.50 \left(\frac{M_*}{10^{11} M_\odot}\right) \tag{4}$$
- $V_{\rm vir}$ is the virial velocity
- T is the time elapsed between the two mergers, i.e. $T_{\rm next\_merger}$

M_*, $V_{\rm vir}$ and $M_{\rm cold}$ are taken from the simulation for the resulting galaxy with m_1 and m_2 as progenitors of the central black hole, to be formed from the coalescence of $m_1 + m_2$.
In the following histogram we plot the ratio between the mass given by the Millennium Simulation and the mass resulting implementing the hot gas accretion above. We analyze each tree, for each black hole in a tree we search for its descendant. So we calculate the mass of the descendant accreting hot gas for the time between the mergers and we compare this mass with that given by the simulation.



We observe that the vast majority of systems have the same mass, either using the hot accretion from the model or the mass given by the simulation. Nevertheless, the simulation gives slightly higher masses.

```
i=0
while(i<len(mass_bh)):
  print(i)
  j=i
  while(mass1[j]!=-1):
    j=j+1
    if(j>=len(mass_bh)):
      break
  tree_index=j
  for k in range(i,tree_index):
    mass_descendant=(lookback.find_mass_descendant(k, tree_index))[0]
    mass_hot=mass_bh[k]+radio(time_to_next_merger[k]*Gyr/t_1yr,hot_mass[k],\
vvir[k],mass_bh[k],stellar_mass[k])

  i=tree_index+1
  if(i>=len(mass_bh)):
    break
```

1.5 Cold gas accretion during merger - quasar mode

When two black holes merge, the mass of the resulting black hole is not quite a direct sum of the progenitor masses (we ignore gravitational wave losses). Instead, during a merger, the central black hole of the major progenitor grows both by absorbing the central black hole of the minor progenitor and by accreting cold gas (quasar mode). The resulting black hole mass is then given by the sum of the two progenitor black holes, m_1 and m_2, plus a δm given by cold gas accretion during the merger (see [Arxiv0508046])

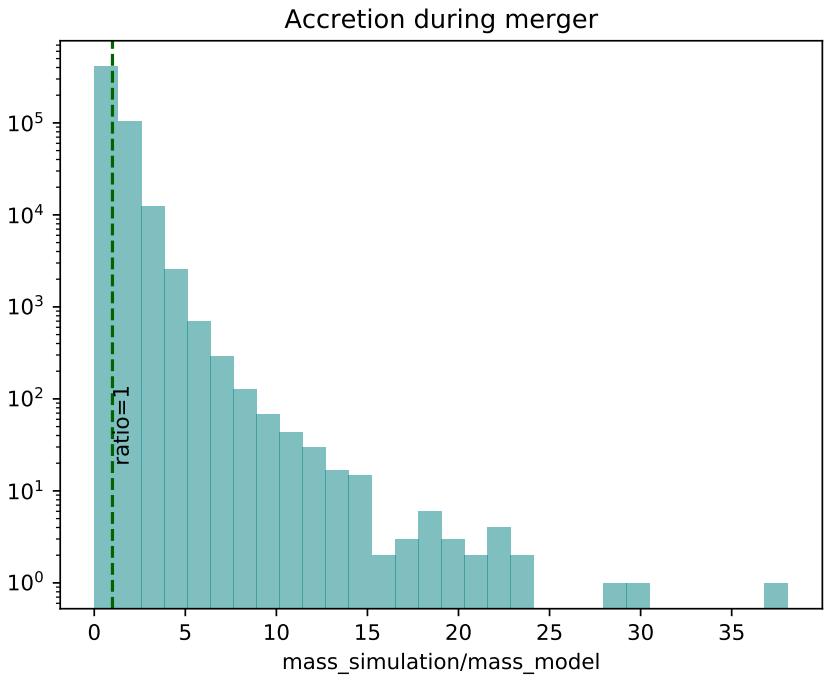
$$\delta m_{\rm quasar\_mode} = f\left(\frac{M_{\rm min}}{M_{\rm maj}}\right)\left[\frac{M_{\rm cold}}{1+\frac{280{\rm kms}^{-1}}{V_{\rm vir}}}\right] \tag{5}$$

where

- $M_{\rm min}$ and $M_{\rm maj}$ are the total baryon masses of the minor and major progenitors
- $M_{\rm cold}$ is the total cold gas
- f is a free parameter set to $f = 0.03$
- $V_{\rm vir}$ is the virial velocity

$M_{\rm cold}$ and $V_{\rm vir}$ are taken from the simulation data for the resulting galaxy.

In the following plot, we compare the mass of the resulting black hole as given by the simulation and the mass calculated using the cold gas accretion model during a merger, i.e. $m_{\rm BH} = m_{\rm progenitor1} + m_{\rm progenitor2} + \delta m_{\rm quasar\_mode}$. In this case as well the two approaches are similar.



1.5.1 Which masses?

At each step of the implementation we use the masses for the resulting black holes given by the Millennium Simulation. For the progenitor binary we implement, before anything else, cold accretion onto single progenitor black holes, using the excess mass resulting between the merged black hole mass and the sum of the progenitor masses as given by the simulation. We favor the accretion of the smaller black hole until eventually it reaches the same mass as the bigger one. At this point the two black holes accrete mass equally.

```
for i in range(len(galaxyId)):
  if (galaxyId[i]!=-1):
    total=P1_BH_mass[i]+P2_BH_mass[i]
    delta_mass=BH_mass[i]-total
    if(P1_BH_mass[i]>P2_BH_mass[i]):
      if(P2_BH_mass[i]+delta_mass<P1_BH_mass[i]):
        mass1[i]=P1_BH_mass[i]
        mass2[i]=P2_BH_mass[i]+delta_mass
      else:
        mass1[i]=BH_mass[i]/2
        mass2[i]=BH_mass[i]/2
    else:
      if(P1_BH_mass[i]+delta_mass<P2_BH_mass[i]):
        mass1[i]=P1_BH_mass[i]+delta_mass
        mass2[i]=P2_BH_mass[i]
      else:
        mass1[i]=BH_mass[i]/2
        mass2[i]=BH_mass[i]/2
```

To take into account the possibility that a certain binary doesn't coalesce before the subsequent galactic merger, so that at least one of the progenitors is a binary itself, we create auxiliary mass vectors, which consider the mass of each progenitor split between the black holes of the binary, in a mass proportional way. In other words, for example, if $mass1$ is the mass of the first progenitor after the implementation as above, but the first progenitor is still a binary from

a previous merger, then the excess mass between $mass1$ and the total mass of the binary is split between the black holes of the binary maintaining the original mass ratio.

1.6 Calculating merger rates

Merger rates are computed as

$$\frac{dN}{dt_{\rm r}} = \int dz \int d\log(\mathcal{M}) \frac{d^2N}{dz d\log(\mathcal{M})} \frac{1}{V_{\rm sim}} \frac{dV_{\rm c}}{dz} \frac{dz}{dt_{\rm L}} \frac{dt_{\rm L}}{dt_{\rm r}} \tag{6}$$

where $\mathcal{M}$ is the chirp mass[4], $V_{\rm sim}$ is the volume of the simulation,

$$V_{\rm vol} = (500/h){\rm Mpc}^3$$

with $h = 0.73$.
z is the redshift, $V_{\rm c}$ the comoving volume, $t_{\rm L}$ the lookback time and $t_{\rm r}$ the rest frame time. We have the following dependencies:

$$\frac{dV_{\rm c}}{dz} = 4\pi D_{\rm H} \frac{(1+z)^2 D_{\rm A}^2}{E(z)} \tag{7}$$

where $E(z)$ is as defined in (2), $D_{\rm H} = c/H_0$ and $D_{\rm A}$ is the angular diameter distance, which is related to the comoving distance in the following way

$$D_{\rm A} = \frac{D_{\rm C}}{1+z} \tag{8}$$

and

$$D_{\rm C} = D_{\rm H} \int_0^z \frac{dz'}{E(z')} \tag{9}$$

Then

$$\frac{dz}{dt_{\rm L}} = H_0(1+z)E(z) \tag{10}$$

and

$$\frac{dt_{\rm L}}{dt_{\rm r}} = 1+z \tag{11}$$

In the implementation we have used a binning of 0.5 for the redshift and of 0.2 in the logarithmic chirp mass.

[4]The chirp mass is defined, give m_1 and m_2 as

$$\mathcal{M} = \frac{(m_1 m_2)^{3/5}}{(m_1+m_2)^{1/5}}$$

A Bash scripts

A.1 Get_Present_Galaxies_Data

```
#!/bin/bash

DIRNAME=PresentGalaxies
mkdir $DIRNAME

OUTPUTFILE="$DIRNAME/Data.csv";

wget --http-user=asesana --http-passwd=APp0w76 --cookies=on --keep-session-
    cookies --save-cookies=cookie.txt --load-cookies=cookie.txt -O $OUTPUTFILE "
    http://gavo.mpa-garching.mpg.de/MyMillennium?action=doQuery&SQL=
select D.galaxyId,
       D.lastProgenitorId

  from MPAGalaxies..DeLucia2006a D

  where D.blackHoleMass %3E %3D 0.0073
   and  D.snapnum %3D 63

  order by D.galaxyId asc
";
```

A.2 Get_Trees_Data

```
#!/bin/bash

DIRNAME=BHMergerTrees
mkdir $DIRNAME

while read f1 f2;
do OUTPUTFILE="$DIRNAME/Data$f1.csv";

wget --http-user=asesana --http-passwd=APp0w76 --cookies=on --keep-session-
    cookies --save-cookies=cookie.txt --load-cookies=cookie.txt -O $OUTPUTFILE "
    http://gavo.mpa-garching.mpg.de/MyMillennium?action=doQuery&SQL=
select D.galaxyId,
       D.snapnum,
       D.descendantId,
       P1.galaxyId as P1_Id,
       P2.galaxyId as P2_Id,
       D.redshift as D_z,
       D.stellarMass as D_mass,
       D.bulgeMass as D_bulge,
       D.sfr as sfr,
       D.blackHoleMass as D_BH,
       P1.redshift as P1_z,
       P2.redshift as P2_z,
       P1.blackHoleMass as M1,
       P2.blackHoleMass as M2,
       P1.bulgeMass as P1_bulge,
       P2.bulgeMass as P2_bulge,
       P1.stellarMass as P1_stars,
       P2.stellarMass as P2_stars,
```

```
28          D.coldGas as M_cold,
            D.hotGas as M_hot,
30          D.vvir as V_vir

32    from MPAGalaxies..DeLucia2006a D,
            MPAGalaxies..DeLucia2006a P1,
34          MPAGalaxies..DeLucia2006a P2

36    where D.GalaxyId between $f1 and $f2
        and P1.snapnum=P2.snapnum
38      and P1.galaxyId %3C P2.galaxyId
        and P1.descendantId %3D D.galaxyId
40      and P2.descendantId %3D D.galaxyId
        and P1.blackHoleMass %3E 0
42      and P2.blackHoleMass %3E 0
        and P1.bulgeMass %3E 0
44      and P2.bulgeMass %3E 0
        and P1.stellarMass %3E 0
46      and P2.stellarMass %3E 0

48    order by D.snapnum asc
"';
50  done<Data/PresentGalaxiesData.csv
```