CATAM 14.5 Cosmological Distances

1 Distances and Time

1.1 Lookback Time

The lookback time t_L is the difference between the current age of the universe and the time at which a photon was emitted.

$$t_L = t_H \int_0^z \frac{dz'}{(1+z')E(z')} \tag{1}$$

With t_H the Hubble time and $E(z) = \sqrt{\Omega_m(1+z)^3 + \Omega_k(1+z)^2 + \Omega_\lambda}$

Considering the Einstein-de-Sitter model Universe ($\Omega_m = 1,\ \Omega_\lambda = 0$), we can easily compute the age of the Universe.

$$t_0 = t_H \int_0^\infty (1+z')^{-\frac{5}{2}} dz' = -\frac{2}{3} t_H (1+z')^{-\frac{3}{2}} \Big|_0^\infty = \frac{2}{3} t_H$$

1.1.1 Question 2

We will consider four distinct models of the universe:

- Einstein-de-Sitter universe ($\Omega_\mathrm{m} = 1,\ \Omega_\Lambda = 0$)
- Classical closed universe ($\Omega_\mathrm{m} = 2,\ \Omega_\Lambda = 0$)
- Baryon dominated low density universe ($\Omega_\mathrm{m} = 0.04,\ \Omega_\Lambda = 0$)
- Currently popular universe ($\Omega_\mathrm{m} = 0.27,\ \Omega_\Lambda = 0.73$)

A program has been written to tabulate and graph results for lookback times in each of these models. It is called ***Q2.py*** and can be found in Section 3.

Redshift	Einstein-de-Sitter	Classical closed	Baryon Dominated	Currently Popular
0.1	12.06762	11.80719	12.34083	12.68661
1.0	58.55965	52.94232	67.42440	76.28055
2.0	73.15351	64.55459	89.44977	101.87216
4.0	82.48464	71.64913	106.62361	119.34947
6.7	86.34735	74.50169	115.30207	126.735608
10000	90.87743	77.90534	128.53490	135.05773

Table 1: Lookback times in 100 million years for various redshifts, assuming $H_0 = 72\mathrm{kms}^{-1}\mathrm{Mpc}^{-1}$.

Note that in the last entry the maximum redshift is so high that the tail of the integral in equation 1 can be safely ignored. As such this gives a close approximation for the age of the universe.

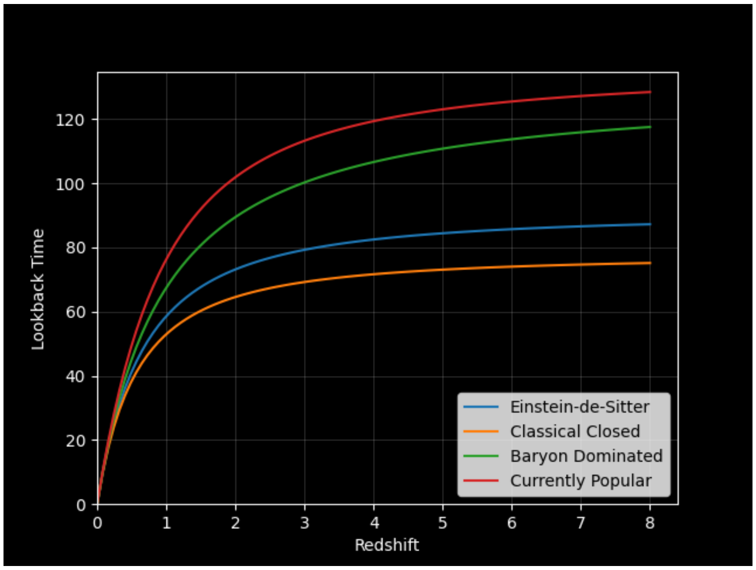


Figure 1: Lookback time graphed against redshift for various universes

We observe in Figure 1 that the lookback time is an increasing function with respect to redshift and plateaus as $z \to \infty$, so we can argue that it represents the age of the universe.

1.2 Distance Measures

Considering the Einstein-de-Sitter model Universe ($\Omega_k = 0$), we get an expression for D_A in:

$$D_A = \frac{D_C}{1+z} = \frac{D_H}{1+z}\int_0^z \frac{dz'}{E(z')} = \frac{D_H}{1+z}\int_0^z (1+z')^{-\frac{3}{2}}dz' = \frac{2D_H}{1+z} - \frac{2D_H}{(1+z)^{\frac{3}{2}}}$$

Finding stationary points:

$$\frac{dD_A}{dz} = -\frac{2D_H}{(1+z)^2} + \frac{3D_H}{(1+z)^{\frac{5}{2}}} = 0$$

$$\implies 2(1+z)^{\frac{1}{2}} = 3 \implies 4 + 4z = 9 \implies z = 1.25$$

Proving it is a maxima:

$$\frac{d^2D_A}{dz^2} = \frac{4D_H}{(1+z)^3} - \frac{15D_H}{2(1+z)^{\frac{7}{2}}}$$

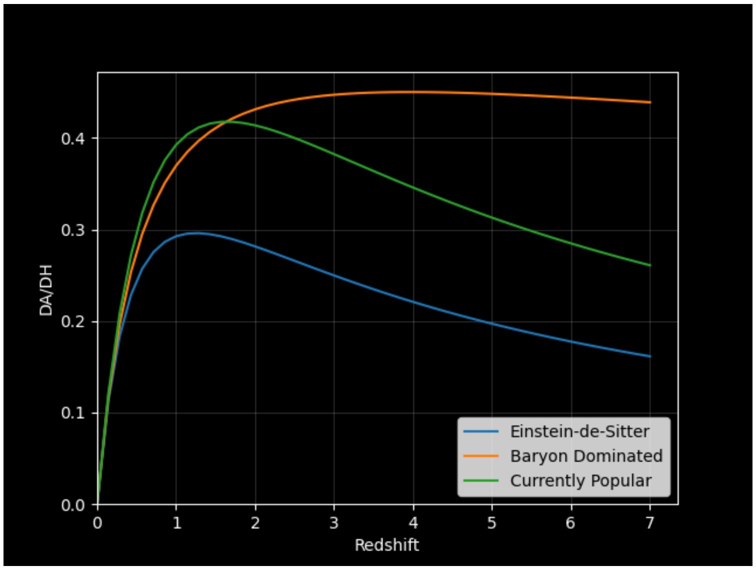
$$\left.\frac{d^2D_A}{dz^2}\right|_{z=1.25} = D_H\left(\frac{256}{729} - \frac{320}{729}\right) = -D_H\frac{64}{729} < 0$$

1.2.1 Question 4

A program has been written to compute D_A and D_L given a redshift and universe model. It is also designed to plot the values of D_A/D_H and D_L/D_H against redshift. It is called ***Q4.py*** and can be found in Section 3.

	Einstein		Baryon		Popular	
Redshift	D_A/D_H	D_L/D_H	D_A/D_H	D_L/D_H	D_A/D_H	D_L/D_H
1.0	0.29260	1.17040	0.36969	1.47874	0.39244	1.56976
1.25	0.29600	1.49850	0.39401	1.99467	0.40981	2.07465
2.0	0.28148	2.53336	0.43112	3.88008	0.41366	3.72297
4.0	0.22089	5.52235	0.45001	11.25031	0.34575	8.64375

Table 2: Distance ratios at certain redshifts for each model universe.



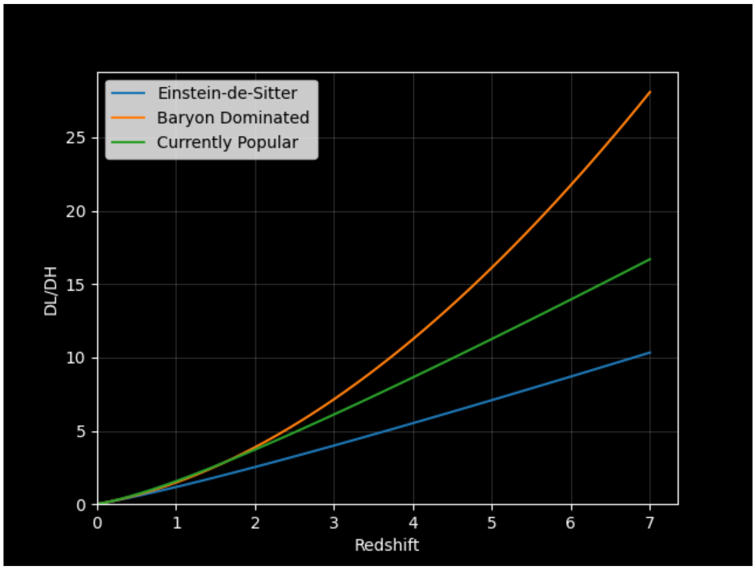


Figure 2: Plots of D_A/D_H and D_L/D_H against redshift

2 Comoving Volume

2.1 Uniform Comoving Distribution

In order to find $\langle V/V_{max}\rangle$ we must average the population weighted according to V/V_{max}. It is clear that we should use the given formula for the number of objects to produce our weighting. Assuming we have a uniform distribution of the number of cosmological objects over the range $L \in (0, \infty), V \in (0, V_{max}(L))$, this yields the following PDF ϕ:

$$\phi(V, L) = \frac{1}{\int_0^\infty \Phi(L) \int_0^{V_{max}(L)} dV dL}$$

Then computing $\langle V/V_{max}\rangle$ is simply a matter of computing the expectation $\mathbb{E}(V/V_{max})$.

$$\langle V/V_{max}\rangle \;=\; \frac{\int_0^\infty \Phi(L) \int_0^{V_{max(L)}} V/V_{max} dV dL}{\int_0^\infty \Phi(L) \int_0^{V_{max}(L)} dV dL} = \frac{\int_0^\infty \Phi(L)\frac{1}{2}V_{max} dL}{\int_0^\infty \Phi(L) V_{max} dL} = \frac{1}{2}$$

2.2 Computing V/V_{max}

2.2.1 Question 6

A program has been written to compute V and V_{max} for an arbitrary redshift z and a ratio between detected and minimum flux f/f_0. It works by computing the maximum redshift z_0 corresponding to a flux of f_0 by way of binary search; a sufficiently fast algorithm. Then using z and z_0 along with the formulas provided to compute D_C in each case, yielding V and V_{max}. The program is called ***Q6.py*** and can be found in Section 3.

In an extension to this program we generate random numbers for Ω_m, z and f/f_0 with z small. When we plot $(f/f_0)^{-\frac{3}{2}}$ against V/V_{max}, it is clear from Figure 3 that, in the Euclidean limit, they are directly proportional.

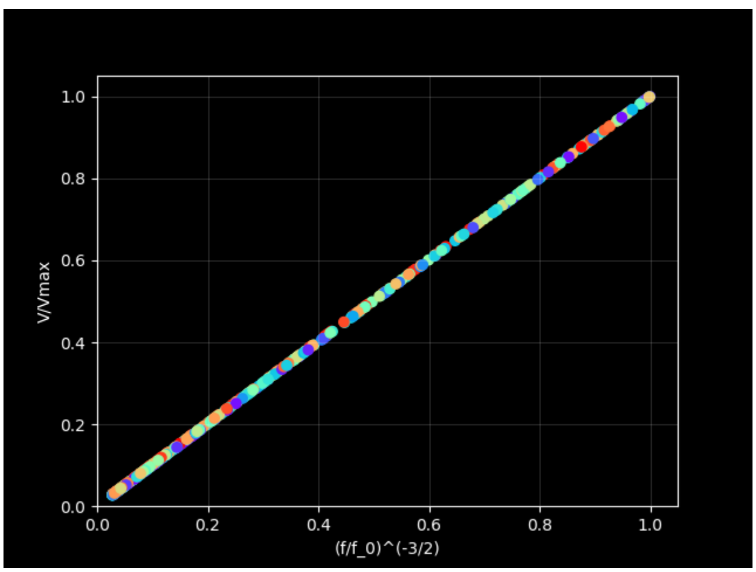


Figure 3: Plot of $(f/f_0)^{-\frac{3}{2}}$ against V/V_{max}

Computing the value of $\langle V/V_{max}\rangle$ for the popular universe ($\Omega_m = 0.27, \Omega_\lambda = 0.73$) using the quasar dataset provided, we attained a value of 0.7060645570576465. This is much higher than expected from a constant comoving population, which may be motivated by the following two explanations:

- Quasars in the early universe have higher luminosity in the early universe.
- There were a greater density of quasars when the universe was younger.

2.2.2 Question 7

For all the quasars in the dataset $z \in (0.2, 3)$, as this is the range at which an object can be recognised as a quasar. This means there is a maximum volume of space that is relevant for consideration. It makes sense then to consider V as the volume closer to the observer than the detected quasar, excluding the space $z < 0.2$. Also along the same lines, we consider V_{max} to be the maximum volume possible with the same luminosity L at minimum flux f_0, excluding the space $z < 0.2$ and $z > 3$.

Formulating this as a probability distribution as before, first let V_0, V_1 be the volume of space with $z < 0.2$ and $z < 3$ respectively. We attain the following uniform PDF as before:

$$\frac{1}{\int_0^\infty \Phi(L) \int_0^{V_{max}(L)} dV dL}$$

But this time over the ranges $L \in (0, \infty), V \in (0, V_{max}(L)) \subseteq (0, V_1 - V_0)$.

Calculating $\langle V/V_{max} \rangle$ it is the same as before yielding $\frac{1}{2}$:

$$\langle V/V_{max} \rangle = \frac{\int_0^\infty \Phi(L) \int_0^{V_{max}(L)} V/V_{max} dV dL}{\int_0^\infty \Phi(L) \int_0^{V_{max}(L)} dV dL} = \frac{\int_0^\infty \Phi(L) \frac{1}{2} V_{max} dL}{\int_0^\infty \Phi(L) V_{max} dL} = \frac{1}{2}$$

Q7.py was written to run this edit on the quasar data. It is found in Section 3.

Running this program we attain a value of 0.689145751083584 for $\langle V/V_{max} \rangle$, which is indeed closer to the expected value of 0.5, though still significantly overshoots, again suggesting some combination of higher luminosity and greater density of quasars in the early universe.

3 Code

3.1 Q2.py

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

def SetupAxes (x, y, showGrid=True):

    x = np.array(x)
    y = np.array(y)
    xMax = np.max(x)
    xMin = 0
    yMax = np.max(y)
    yMin = 0
    dx = (xMax - xMin) / 20
    dy = (yMax - yMin) / 20

    fig, ax = plt.subplots()
    plt.xlabel('Redshift', color='white')
    plt.ylabel('Lookback Time', color='white')
    fig.set_facecolor("black")
    ax.set_facecolor("black")
    ax.spines['bottom'].set_color('white')
    ax.spines['top'].set_color('white')
    ax.spines['left'].set_color('white')
    ax.spines['right'].set_color('white')
    ax.tick_params(colors='white')
    ax.set_xlim(xMin, xMax+dx)
    ax.set_ylim(yMin, yMax+dy)
    if (showGrid):
        ax.grid(color='white', linewidth=0.4, alpha=0.3, zorder=0)
    return fig, ax


def ComputeLookbackTime (z, h = 0.72, Omega_m = 0.27, Omega_Lambda = 0.73, n = 10000):
    Omega_k = 1 - Omega_m - Omega_Lambda
    tH = 3.0856 * 10**17 / (10**8 * 60 * 60 * 24 * 365 * h)

    def E(x):
        return np.sqrt(Omega_m * np.power((1+x),3) + Omega_k * np.power(1+x, 2) + Omega_Lambda)

    x = np.linspace(0, z, n)
    y = 1/ ((x + 1) * E(x))
    integral = tH * np.trapz(y, dx=z/n)
    return integral





zs = np.array([0.1, 1.0, 2.0, 4.0, 6.7])
times =[[0 for j in range(5)] for i in range(4)]
```

```python
for i in range(5):
    times[0][i] = ComputeLookbackTime(zs[i], Omega_m = 1, Omega_Lambda = 0)
    times[1][i] = ComputeLookbackTime(zs[i], Omega_m = 2, Omega_Lambda = 0)
    times[2][i] = ComputeLookbackTime(zs[i], Omega_m = 0.04, Omega_Lambda = 0)
    times[3][i] = ComputeLookbackTime(zs[i], Omega_m = 0.27, Omega_Lambda = 0.73)
print(times)

n = 100
zs = np.linspace(0, 8, n)
times =[[0 for j in range(n)] for i in range(4)]


for i in range(n):
    times[0][i] = ComputeLookbackTime(zs[i], Omega_m = 1, Omega_Lambda = 0)
    times[1][i] = ComputeLookbackTime(zs[i], Omega_m = 2, Omega_Lambda = 0)
    times[2][i] = ComputeLookbackTime(zs[i], Omega_m = 0.04, Omega_Lambda = 0)
    times[3][i] = ComputeLookbackTime(zs[i], Omega_m = 0.27, Omega_Lambda = 0.73)


fig, ax = SetupAxes(zs, times)

for i in range(4):
    plt.plot(zs, times[i])
ax.legend(['Einstein-de-Sitter', 'Classical Closed', 'Baryon Dominated', 'Currently Popular'])
plt.show()

print(ComputeLookbackTime(10000, Omega_m = 1, Omega_Lambda = 0, n= 100000))
print(ComputeLookbackTime(10000, Omega_m = 2, Omega_Lambda = 0, n= 100000))
print(ComputeLookbackTime(10000, Omega_m = 0.04, Omega_Lambda = 0, n= 100000))
print(ComputeLookbackTime(10000, Omega_m = 0.27, Omega_Lambda = 0.73, n= 100000))
```

3.2 Q4.py

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl


def SetupAxes (x, y, ytext = 'amogus', showGrid=True):

    x = np.array(x)
    y = np.array(y)
    xMax = np.max(x)
    xMin = 0
    yMax = np.max(y)
    yMin = 0
    dx = (xMax - xMin) / 20
    dy = (yMax - yMin) / 20

    fig, ax = plt.subplots()
    plt.xlabel('Redshift', color='white')
    plt.ylabel(ytext, color='white')
    fig.set_facecolor("black")
    ax.set_facecolor("black")
    ax.spines['bottom'].set_color('white')
    ax.spines['top'].set_color('white')
    ax.spines['left'].set_color('white')
    ax.spines['right'].set_color('white')
    ax.tick_params(colors='white')
    ax.set_xlim(xMin, xMax+dx)
    ax.set_ylim(yMin, yMax+dy)
    if (showGrid):
        ax.grid(color='white', linewidth=0.4, alpha=0.3, zorder=0)
    return fig, ax


def ComputeValues(z, Omega_m = 0.27, Omega_Lambda = 0.73):
    Omega_k = 1 - Omega_m - Omega_Lambda

    DADH = 0

    def E(x):
        return np.sqrt(Omega_m * np.power((1+x),3) + Omega_k * np.power(1+x, 2) + Omega_Lambda)

    x = np.linspace(0, z, 1000)
    y = 1/ E(x)
    DCDH = np.trapz(y, dx=z * 0.001)
    if (Omega_k == 0):
        DADH = DCDH / (1+z)
    elif (Omega_k > 0):
        DADH = 1/ (np.sqrt(Omega_k) * (1+z)) * np.sinh(np.sqrt(Omega_k) * DCDH)
    else:
        DADH = 1/ (np.sqrt(Omega_k) * (1+z)) * np.sin(np.sqrt(Omega_k) * DCDH)

    DLDH = (1+z)**2 * DADH
```

```python
    return DADH, DLDH

n=50
z = np.array(np.linspace(0, 7, n))

DADHs =[[0 for j in range(n)] for i in range(3)]
DLDHs =[[0 for j in range(n)] for i in range(3)]

for i in range(n):
    DADHs[0][i], DLDHs[0][i] = ComputeValues(z[i], Omega_m = 1, Omega_Lambda = 0)
    DADHs[1][i], DLDHs[1][i] = ComputeValues(z[i], Omega_m = 0.04, Omega_Lambda = 0)
    DADHs[2][i], DLDHs[2][i] = ComputeValues(z[i], Omega_m = 0.27, Omega_Lambda = 0.73)

fig1, ax1 = SetupAxes(z, DADHs, ytext='DA/DH')
for i in range(3):
    plt.plot(z, DADHs[i])
ax1.legend(['Einstein-de-Sitter', 'Baryon Dominated', 'Currently Popular'])

fig2, ax2 = SetupAxes(z, DLDHs, ytext='DL/DH')
for i in range(3):
    plt.plot(z, DLDHs[i])
ax2.legend(['Einstein-de-Sitter', 'Baryon Dominated', 'Currently Popular'])
plt.show()

z = [1, 1.25, 2, 4]

DADHs =[[0 for j in range(4)] for i in range(3)]
DLDHs =[[0 for j in range(4)] for i in range(3)]

for i in range(4):
    DADHs[0][i], DLDHs[0][i] = ComputeValues(z[i], Omega_m = 1, Omega_Lambda = 0)
    DADHs[1][i], DLDHs[1][i] = ComputeValues(z[i], Omega_m = 0.04, Omega_Lambda = 0)
    DADHs[2][i], DLDHs[2][i] = ComputeValues(z[i], Omega_m = 0.27, Omega_Lambda = 0.73)

for i in range(3):
    print(DADHs[i])
    print(DLDHs[i])
    print("\n")
```

3.3 Q6.py

```python
import numpy as np
import csv
import matplotlib as mpl
import matplotlib.pyplot as plt

def SetupAxes (x, y, showGrid=True):

    x = np.array(x)
    y = np.array(y)
    xMax = np.max(x)
    xMin = 0
    yMax = np.max(y)
    yMin = 0
    dx = (xMax - xMin) / 20
    dy = (yMax - yMin) / 20

    fig, ax = plt.subplots()
    plt.xlabel('(f/f_0)^(-3/2)', color='white')
    plt.ylabel('V/Vmax', color='white')
    fig.set_facecolor("black")
    ax.set_facecolor("black")
    ax.spines['bottom'].set_color('white')
    ax.spines['top'].set_color('white')
    ax.spines['left'].set_color('white')
    ax.spines['right'].set_color('white')
    ax.tick_params(colors='white')
    ax.set_xlim(xMin, xMax+dx)
    ax.set_ylim(yMin, yMax+dy)
    if (showGrid):
        ax.grid(color='white', linewidth=0.4, alpha=0.3, zorder=0)
    return fig, ax

def ComputeValues(z, Omega_m = 0.27, Omega_Lambda = 0.73):
    Omega_k = 1 - Omega_m - Omega_Lambda

    DADH = 0

    def E(x):
        return np.sqrt(Omega_m * np.power((1+x),3) + Omega_k * np.power(1+x, 2) + Omega_Lambda)

    x = np.linspace(0, z, 1000)
    y = 1/ E(x)
    DCDH = np.trapz(y, dx=z * 0.001)
    if (Omega_k == 0):
        DADH = DCDH / (1+z)
    elif (Omega_k > 0):
        DADH = 1/ (np.sqrt(Omega_k) * (1+z)) * np.sinh(np.sqrt(Omega_k) * DCDH)
    else:
        DADH = 1/ (np.sqrt(Omega_k) * (1+z)) * np.sin(np.sqrt(Omega_k) * DCDH)

    DLDH = (1+z)**2 * DADH
    return DCDH, DADH, DLDH
```

```
def GetVolumes(z, ff0, h = 0.72, Omega_m = 0.27, Omega_Lambda = 0.73):
    # Using Giga-Lightyears
    DH = h * 9.26 * 1.057

    DCDH, DADH, DLDH = ComputeValues(z, Omega_m=Omega_m, Omega_Lambda=Omega_Lambda)
    DC = DH * DCDH
    DL = DH * DLDH

    V = 4 * np.pi / 3 * DC**3

    DL0 = DL * np.sqrt(ff0)


    def E(x):
        return np.sqrt(Omega_m * np.power((1+x),3) + 0 * np.power(1+x, 2) + Omega_Lambda)

    # Find z0 (or z_max)

    # Use a bit of root finding methods to get z0

    a = DL0 / DH
    def BinarySearch(zmin, zmax):
        zmid = (zmax + zmin) / 2
        if (zmax - zmin < 10**-8): return zmid

        x = np.linspace(0,zmid, 1000)
        y = 1/E(x)
        b = (1+zmid) * np.trapz(y, dx = zmid/1000)

        if (a < b):
            return BinarySearch(zmin, zmid)
        elif (a > b):
            return BinarySearch(zmid, zmax)
        else:
            return zmid

    z0 = BinarySearch(z, 20)

    DC0DH, DA0DH, DL0DH = ComputeValues(z0, Omega_m=Omega_m, Omega_Lambda=Omega_Lambda)
    DC0 = DH * DC0DH

    Vmax = 4 * np.pi / 3 * DC0**3

    return  V, Vmax


def SmallRedShift():
    n = 500
    z = [np.random.rand() * 0.01 for i in range(n)]
    ff0 = [(1 + (np.random.rand() ** 2)*10) for i in range(n)]
    Omega_m = [np.random.rand() for i in range(n)]
    VVmax = [0 for i in range(n)]
```

```
    xData = [ff0[i] ** (-1.5) for i in range(n)]
    colours = [mpl.cm.rainbow(z[i]*100) for i in range(n)]

    for i in range(n):
        V, Vmax = GetVolumes(z[i], ff0[i], Omega_m=Omega_m[i], Omega_Lambda=(1-Omega_m[i]))
        VVmax[i] = V/Vmax


    fig, ax = SetupAxes(xData, VVmax)

    plt.scatter(xData, VVmax, c=colours)

    plt.show()

def AveragePerUniverse(n, z, ff0, Omega_m):
    Omega_Lambda = 1 - Omega_m

    tot = 0
    for i in range(n):
        V, Vmax = GetVolumes(z[i], ff0[i], Omega_m=Omega_m, Omega_Lambda=Omega_Lambda)
        VVmax = V/Vmax
        tot += VVmax

    avg = tot/n
    return avg



if __name__ == "__main__":
    SmallRedShift()

    n = 1

    z = [5]
    ff0 = [10]
    avg = AveragePerUniverse(n, z, ff0, 1)


    n = 114

    z = []
    ff0 = []
    with open("quasar.csv", 'r') as file:
        csvreader = csv.reader(file)

        for row in csvreader:
            z.append(float(row[0]))
            ff0.append(float(row[1]))

    n = len(z)

    avgEin = AveragePerUniverse(n, z, ff0, 1)
    avgPop = AveragePerUniverse(n, z, ff0, 0.27)
    print("Einstein De-Sitter: ", avgEin, "Popular: ", avgPop)
```

3.4 Q7.py

```
from Q6 import *

def UpdatedGetVolumes(z, ff0, h = 0.72, Omega_m = 0.27, Omega_Lambda = 0.73):
    # Using Giga-Lightyears
    DH = h * 9.26 * 1.057

    DC0DH, DA0DH, DL0DH = ComputeValues(0.2, Omega_m=Omega_m, Omega_Lambda=Omega_Lambda)
    DC0 = DH * DC0DH
    DC1DH, DA1DH, DL1DH = ComputeValues(3, Omega_m=Omega_m, Omega_Lambda=Omega_Lambda)
    DC1 = DH * DC1DH

    V0 = 4 * np.pi / 3 * DC0**3
    V1 = 4 * np.pi / 3 * DC1**3

    V, Vmax = GetVolumes(z, ff0, h = 0.72, Omega_m = Omega_m, Omega_Lambda = Omega_Lambda)
    V -= V0
    Vmax -= V0
    Vmax = min(Vmax, V1 - V0)

    if (V > Vmax):
        print(z, ff0, V, Vmax)

    return V, Vmax


def UpdatedAveragePerUniverse(n, z, ff0, Omega_m):
    Omega_Lambda = 1 - Omega_m

    tot = 0
    for i in range(n):
        V, Vmax = UpdatedGetVolumes(z[i], ff0[i], Omega_m=Omega_m, Omega_Lambda=Omega_Lambda)
        VVmax = V/Vmax
        tot += (VVmax)

    avg = tot/n
    return avg


if __name__ == "__main__":
    n = 114

    z = []
    ff0 = []
    with open("quasar.csv", 'r') as file:
        csvreader = csv.reader(file)

        for row in csvreader:
            z.append(float(row[0]))
            ff0.append(float(row[1]))

    n = len(z)
```

```
avgEin = UpdatedAveragePerUniverse(n, z, ff0, 1)
avgPop = UpdatedAveragePerUniverse(n, z, ff0, 0.27)
print("AVERAGE: ", avgEin, "Popular: ", avgPop)
```