14.5 Cosmological Distances

Lookback Time

Question 1

We start with the dimensionless form of the given Einstein field equation:

$$\Omega_m + \Omega_\Lambda + \Omega_k = 1 \tag{1}$$

For $\Omega_m = 1$ and $\Omega_\Lambda = 0$, we must have $\Omega_k = 0$. This corresponds to the function:

$$E(z) = (1+z)^{-\frac{3}{2}} \tag{2}$$

Substituting this into the lookback time allows for us to solve for t_L.

$$t_L(z) = t_H \int_0^z \frac{dz'}{(1+z')E(z')} \tag{3}$$

$$= t_H \int_0^z (1+z')^{-\frac{5}{2}}\, dz' \tag{4}$$

$$= \frac{2}{3} t_H \left[1 - (1+z)^{-\frac{3}{2}}\right] \tag{5}$$

The age of the universe t_0 can be deduced by setting $z = \infty$, which sets the second term in equation 5 to 0.

$$\therefore\ t_0 = t_L(z=\infty) = \frac{2}{3} t_H \tag{6}$$

Question 2

Here we determine the lookback time for a general H_0, Ω_m and Ω_Λ corresponding to different universes. The *lookback.py* program on page 13 numerically integrates equation 3 giving the results below.

z	t_L (Gyr)
0.1	1.21
1.0	5.85
2.0	7.31
4.0	8.24
6.7	8.63

Table 1: Einstein-de Sitter universe. Lookback time for $(\Omega_m, \Omega_\Lambda) = (1, 0)$ to 2 decimal places.

z	t_L (Gyr)
0.1	1.18
1.0	5.29
2.0	6.45
4.0	7.16
6.7	7.45

Table 2: Classical closed universe. Lookback time for $(\Omega_m, \Omega_\Lambda) = (2, 0)$ to 2 decimal places.

z	t_L (Gyr)
0.1	1.23
1.0	6.74
2.0	8.94
4.0	10.66
6.7	11.52

Table 3: Baryon dominated low density universe. Lookback time for $(\Omega_m, \Omega_\Lambda) = (0.04, 0)$ to 2 decimal places.

z	t_L (Gyr)
0.1	1.27
1.0	7.62
2.0	10.18
4.0	11.93
6.7	12.67

Table 4: Currently popular universe. Lookback time for $(\Omega_m, \Omega_\Lambda) = (0.27, 0.73)$ to 2 decimal places.

To get the age of the universe t_0, we need to compute $t_L(z = \infty)$.

$(\Omega_m, \Omega_\Lambda)$	t_0 (Gyr)
$(1, 0)$	9.0
$(2, 0)$	7.7
$(0.04, 0)$	12.8
(0.27,0.73)	13.5

Table 5: Age of the universe for the various universes to the nearest 100 million years.

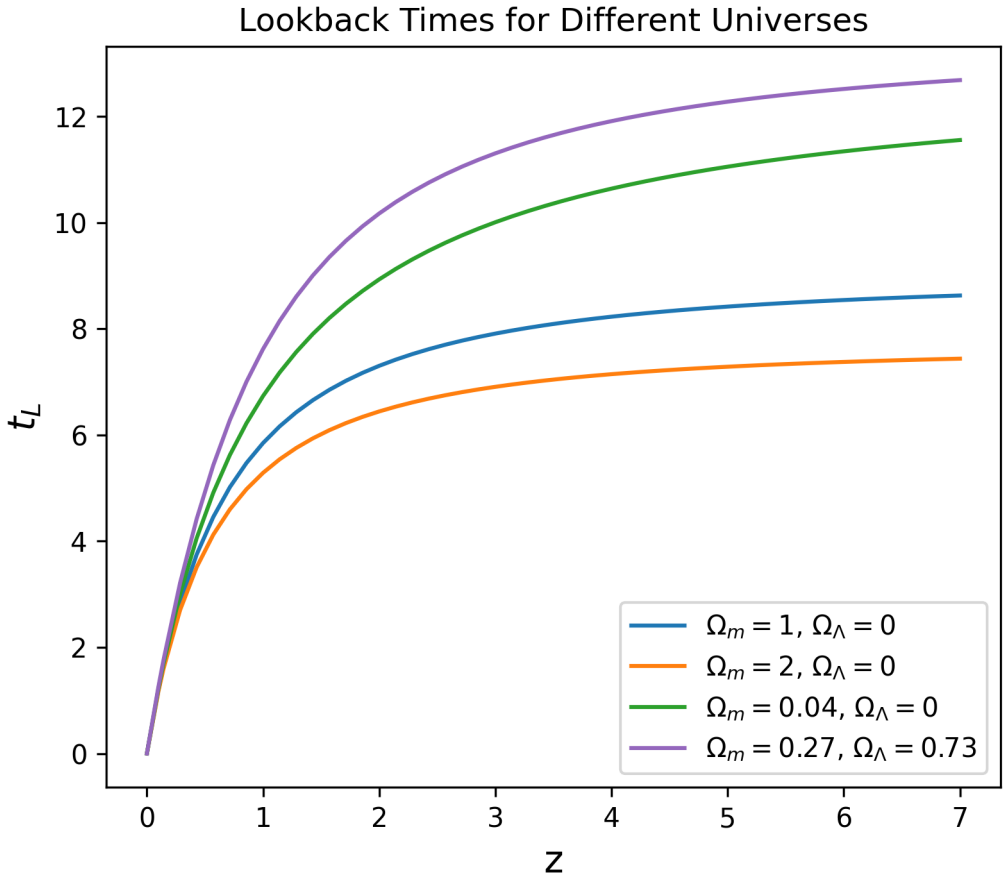

Figure 1: Lookback time plotted against redshift for the 4 different cases.

Figure 1 demonstrates an overall trend that the lookback time tends to a constant as redshift increases to large values - indicating the age of the universe. This makes sense as increasing redshift means we are further back in time and the age of emitted photons t_e is younger. As $t_L = t_0 - t_e$, the lookback time increases at a slower rate with increasing redshift.

Table 5 shows that for universes with no cosmological constant, lower values of Ω_m have greater lookback times - suggesting that more matter results in a younger universe age.

Once Λ is introduced, we record the highest t_L across these 4 universes, despite the currently popular universe having a higher Ω_m than the low density universe. Note this t_L is close to the actaul age of our universe - 13.7 Gyr. This suggests cosmological constant plays a more significant role in the expansion of the universe than matter does.

Distance Measures

Question 3

For the case $\Omega_m = 1$ and $\Omega_\Lambda = 0$, we have $\Omega_k = 0$ and therefore an Einstein-de Sitter universe. The corresponding angular diameter distance for $\Omega_k = 0$ is:

$$D_A = \frac{D_C}{1+z} \tag{7}$$

Substituting in the line of sight comoving distance with the $E(z)$ in equation 2 gives:

$$D_A = \frac{D_H}{1+z} \int_0^z (1+z')^{-\frac{3}{2}}\, dz' \tag{8}$$

$$D_A = 2D_H \left[(1+z)^{-1} - (1+z)^{-\frac{3}{2}}\right] \tag{9}$$

To find the stationary point, we evaluate the derivative when set to 0.

$$\frac{dD_A}{dz} = 2D_H \left[-(1+z)^{-2} + \frac{3}{2}(1+z)^{-\frac{5}{2}}\right] = 0 \tag{10}$$

$$=> \;\; z = 1.25 \tag{11}$$

To check this is a maximum, we evaluate the second derivative of D_A at this redshift.

$$\left.\frac{d^2 D_A}{dz^2}\right|_{z=1.25} = 2D_H \underbrace{\left[2(1+z)^{-3} - \frac{15}{4}(1+z)^{-\frac{7}{2}}\right]}_{<0} \tag{12}$$

As $D_H > 0$, the second derivative at this point is negative, verifying $z = 1.25$ is a maximum.

Question 4

The *distances.py* program on page 15 evaluates the dimensionless angular diameter and luminosity distances for a range of redshifts and different combinations of Ω_m, Ω_Λ and Ω_k. Below are the results of the program for 3 different cases of $(\Omega_m, \Omega_\Lambda)$.

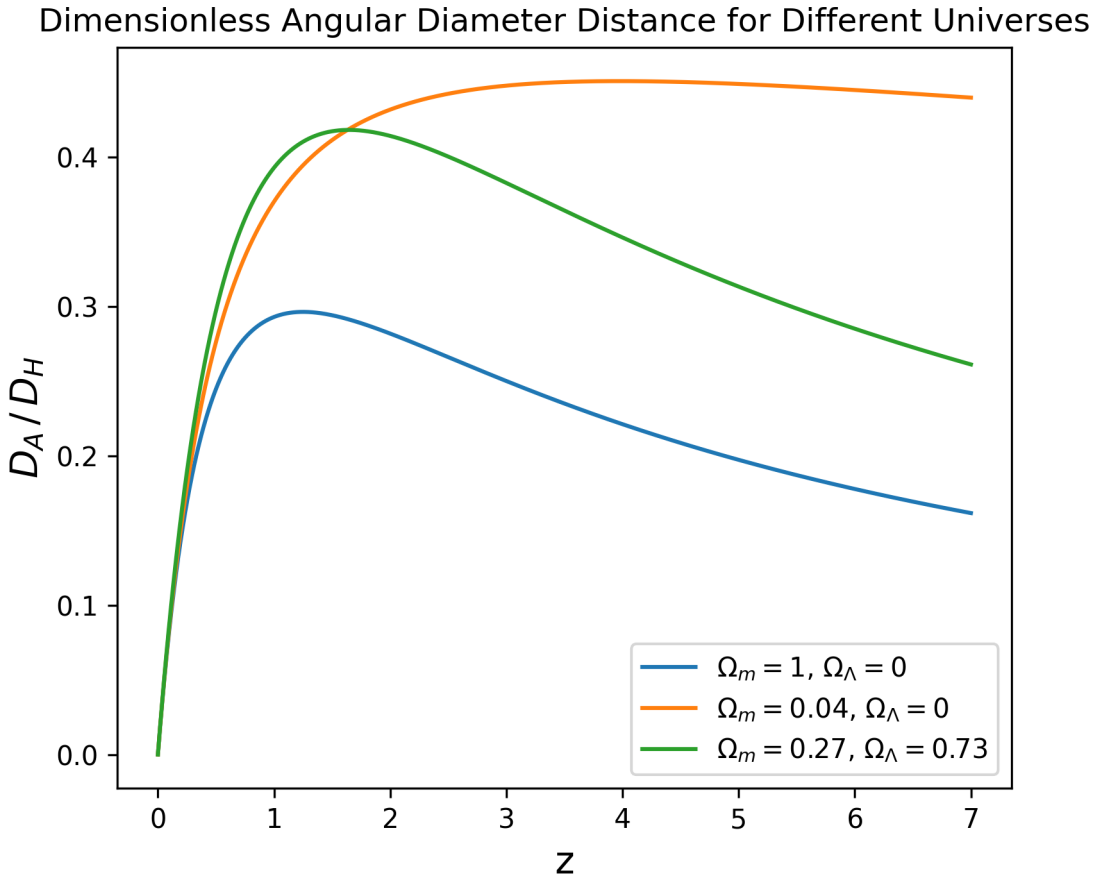

Figure 2: Dimensionless values of D_A for the 3 different cases plotted in the range $0 < z < 7$.

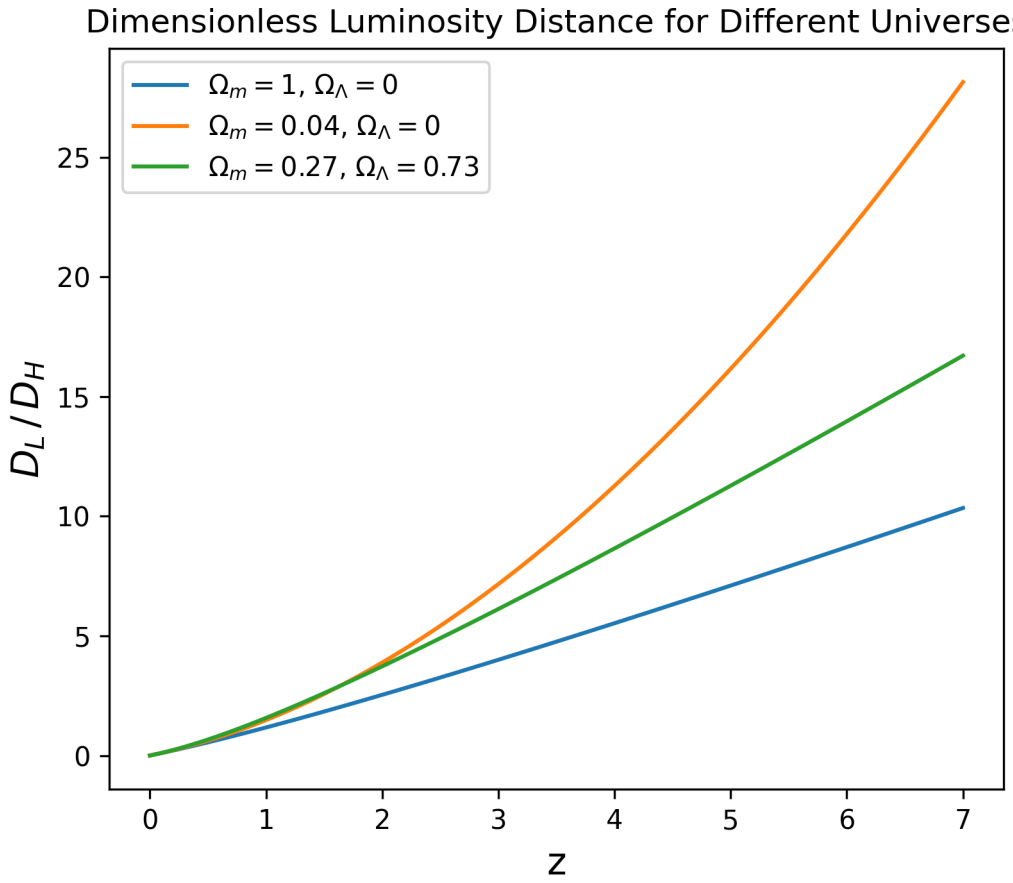

Figure 3: Dimensionless values of D_L for the 3 different cases plotted in the range $0 < z < 7$.

z	D_A/D_H	D_L/D_H
1	0.29	1.17
1.25	0.30	1.50
2.0	0.28	2.54
4.0	0.22	5.53

Table 6: Einstein-de Sitter universe. Luminosity and angular diameter distances for $(\Omega_m, \Omega_\Lambda) = (1, 0)$ to 2 decimal places.

z	D_A/D_H	D_L/D_H
1	0.37	1.48
1.25	0.39	2.00
2.0	0.43	3.89
4.0	0.45	11.27

Table 7: Baryon dominated low density universe. Luminosity and angular diameter distances for $(\Omega_m, \Omega_\Lambda) = (0.04, 0)$ to 2 decimal places.

z	D_A/D_H	D_L/D_H
1	0.39	1.57
1.25	0.41	2.08
2.0	0.41	3.73
4.0	0.35	8.65

Table 8: Currently popular universe. Luminosity and angular diameter distances for $(\Omega_m, \Omega_\Lambda) = (0.27, 0.73)$ to 2 decimal places.

Comoving Volume

Question 5

Let $y = \frac{V}{V_{max}}$ for convenience. The expectation value can be rewritten as an integral over y using a probability density function $P(y)$. The maximum value of y is 1 as $V \leq V_{max}$, so the limits of integration are 0 to 1.

$$\langle y \rangle = \int_0^1 y\, P(y)\, dy \tag{13}$$

For a uniform comoving distribution, we can write $P(y) = A$, where A is a constant. Normalisation requires:

$$\int_0^1 P(y) dy = 1 \tag{14}$$

$$A \int_0^1 dy = 1 \tag{15}$$

$$A = 1 \tag{16}$$

Equation 13 now becomes:

$$\langle y \rangle = \int_0^1 y\, dy = \frac{1}{2} \tag{17}$$

$$\therefore \left\langle \frac{V}{V_{max}} \right\rangle = \frac{1}{2} \tag{18}$$

Question 6

The program *comoving.py* on page 18 reads the *quasar.dat* file and outputs arrays of V, V_{max} and $\left\langle \frac{V}{V_{max}} \right\rangle$ as detailed below.

V can be calculated directly from the sample of z given in the data by using the comoving volume equation with the definition of D_C:

$$V = \frac{4\pi}{3} D_C^3 \tag{19}$$

$$D_C = D_H \int_0^z \frac{dz'}{E(z')} \tag{20}$$

To find V_{max}, we combine the fluxes $f = \frac{L}{4\pi D_L^2}$ and $f_0 = \frac{L}{4\pi D_{L,max}^2}$ to get:

$$\frac{f}{f_0} = \frac{D_{L,max}^2}{D_L^2} \tag{21}$$

Substituting in the relation $D_L = (1+z)D_C$, we get:

$$(1+z_{max}) \int_0^{z_{max}} \frac{dz'}{E(z')} = \sqrt{\frac{f}{f_0}} (1+z) \int_0^z \frac{dz'}{E(z')} \tag{22}$$

The program solves this using the *brentq* function from the *scipy* library to numerically find the root of z_{max} in equation 22, which corresponds to each z and f/f_0.

For the currently popular universe with $\Omega_m = 0.27$ and $\Omega_\Lambda = 0.73$, the program gives the result to 2 significant figures:

$$\left\langle \frac{V}{V_{max}} \right\rangle = 0.71.$$

This is not quite the value expected from a constant comoving population where $\left\langle \frac{V}{V_{max}} \right\rangle = \frac{1}{2}$.

As the result is higher than the expected value, this suggests that this sample of quasars are on average found closer to their $z_{\max}$ than to the observer. So the observed flux of the quasar is closer to f_0. This suggests that these quasars have existed for a long time throughout the expansion of the universe such that its observed flux has lowered.

This non-uniformity obtained suggests this distribution of quasars is not uniform and these objects may be grouped together in clusters at areas.

Verifying the Euclidean limit

In order to verify this limit, we plot the volume ratio against the flux ratio arrays for small z only. If proportional, the graph should present a linear relationship. We can test for this by fitting a line of the form

$$\frac{V}{V_{max}} = \alpha \left(f/f_0\right)^{-\frac{3}{2}} + \beta \tag{23}$$

where α and β are constants to be determined. We expect $\beta = 0$ for the proportionality to hold.

We take the requirement $z < 0.3$ to be sufficiently small relative to the range of z in the *quasar.dat* file.

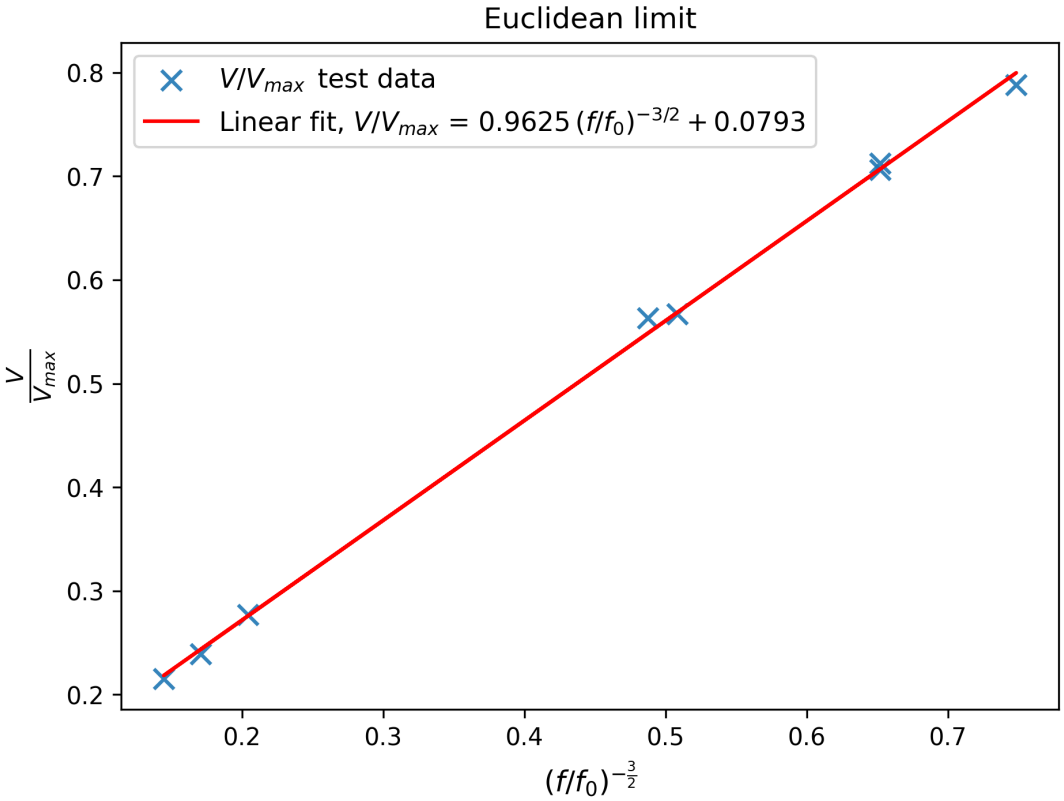

Figure 4: Plot to verify the Euclidean limit by fitting a straight line to the data for $z < 0.3$.

The straight line fit constants are determined to be

$$\alpha = 0.96, \ \beta = 0.08$$

We see that $\beta \approx 0$, so we have indeed verified the Euclidean limit for small z.

$$\frac{V}{V_{max}} = \alpha\,(f/f_0)^{-\frac{3}{2}} + \beta \tag{24}$$

$$\approx \alpha\,(f/f_0)^{-\frac{3}{2}} \tag{25}$$

$$\propto (f/f_0)^{-\frac{3}{2}} \tag{26}$$

Question 7

As the sample is constrained to be within $0.2 < z < 3$, we need to replace the lower limit in the integral for D_C from 0 to 0.2 when considering z and z_{max}.

This gives $\left\langle \frac{V}{V_{max}} \right\rangle = 0.68$.

Programs

lookback.py - used in Question 2

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

# Constants
Mpc = 3.0856e22 # m
H_0 = 72e3 / Mpc # s^-1
t_H = 1 / H_0

def Omega_k(O_m, O_l):
    return 1 - O_m - O_l

def E(z):
    return np.sqrt( (Omega_m * (1+z)**3) + (Omega_k(Omega_m, Omega_l)
       * (1+z)**2) + Omega_l)

def lookback_func(z):
    return t_H * (1 / ((1+z) * E(z)))

def lookback_integral(z):
    integral_sum = 0
    for i in range(1, len(z)):
        integral_sum += lookback_func(z[i]) * (z[i] - z[i-1])
    return integral_sum

def lookback_time(z):
    t_L = np.zeros_like(Z)
    for integral_value,i in enumerate(Z):
        t_L[integral_value] = lookback_integral(z)[integral_value]
    return np.array(t_L) / (3.1556926e7 * 1e9) # converting to Gyr

values = np.linspace(0,7,50) # upper limits of integral to give
   lookback time

b = [0.1, 1.0, 2.0, 4.0, 6.7]
total_arr = np.append(values,b)
Z = np.sort(total_arr)

x = np.linspace(0,Z,2000) # change accuracy via number of steps


Omega_m_values = [1, 2, 0.04, 0.27]
Omega_l_values = [0, 0, 0, 0.73]

universes = ('EDS', 'classical closed', 'baryon', 'currently popular')

all_times = []

plt.figure(figsize = (6,5))
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#9467bd']
```

```
for i in range(0,4):
    Omega_m = Omega_m_values[i]
    Omega_l = Omega_l_values[i]
    all_times.append(lookback_time(x))
    print(f'{universes[i]}: {lookback_time(x)}')
    print('\n')


    plt.plot(Z, all_times[i], color=colors[i], label=f'$\Omega_m={
       Omega_m}$, $\Omega_\Lambda={Omega_l}$')


plt.xlabel('z', fontsize = 14)
plt.ylabel('$t_L$', fontsize = 14)
plt.title('Lookback times for different universes')
plt.legend()
# plt.savefig('Lookback Times Plot.png', dpi=300)
plt.show()

Z_values = [0.1, 1.0, 2.0, 4.0, 6.7]

for l,k in enumerate(all_times):
    for z_val in Z_values:
        print(f'Time {l+1} for z = {z_val}')

        for i,j in enumerate(Z):
            if round(j,2) == z_val:
                print(f'{k[i]:.2f}')
        print('\n')
    print('--------------------------------')
```

distances.py - used in Question 4

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

# Constants
Mpc = 3.0856e22 # m
c = 2.998e8

H_0 = 72e3 / Mpc # s^-1
t_H = 1 / H_0
D_H = c / H_0 # Hubble distance

def Omega_k(O_m, O_l):
    return 1 - O_m - O_l

def E(z):
    return np.sqrt( (Omega_m * (1+z)**3) + (Omega_k(Omega_m, Omega_l)
       * (1+z)**2) + Omega_l)

def distance_func(z):
    return D_H * (1 / E(z))

def midpoint_rule(func, a, b, n):
    h = (b - a) / n
    integral = 0
    for i in range(n):
        midpoint = a + (i + 0.5) * h
        integral += func(midpoint)
    integral *= h
    return integral

def distance_integral(z):  # this is D_C
    return midpoint_rule(distance_func, 0, z, 1000)

def dimensionless_angular(z):
    O_k =  Omega_k(Omega_m,Omega_l)
    if O_k > 0:
        da_over_dh = (1 / (np.sqrt(O_k) * (1+z))) * np.sinh(np.sqrt(
           O_k) * distance_integral(z) / D_H)

    elif O_k == 0:
        da_over_dh = distance_integral(z) / (D_H * (1+z))

    elif O_k < 0:
        da_over_dh = (1 / (np.sqrt(abs(O_k)) * (1+z))) * np.sin(np.
           sqrt(abs(O_k)) * distance_integral(z) / D_H)

    else:
        print('error')
    return da_over_dh

def dimensionless_luminosity(z):
```

```
    DL_over_DH = (1 + z)**2 * dimensionless_angular(z)
    return DL_over_DH

Z = np.linspace(0,7,10000) # upper limits of integral 'z', needs to be
    0<z<7

Omega_m_values = [1, 0.04, 0.27]
Omega_l_values = [0, 0, 0.73]

ang_distances = []
lum_distances = []

fig, ax1 = plt.subplots(figsize=(6,5))
fig, ax2 = plt.subplots(figsize=(6,5))
colors = ['#1f77b4', '#ff7f0e', '#2ca02c']

for i in range(0,3):
    Omega_m = Omega_m_values[i]
    Omega_l = Omega_l_values[i]
    DA_over_DH = dimensionless_angular(Z)
    DL_over_DH = dimensionless_luminosity(Z)

    ang_distances.append(DA_over_DH)
    lum_distances.append(DL_over_DH)

    ax1.plot(Z, DA_over_DH,color = colors[i], linestyle = '-', label=f
       '$\Omega_m={Omega_m}$, $\Omega_\Lambda={Omega_l}$')
    ax2.plot(Z, DL_over_DH, color = colors[i], linestyle = '-', label=
       f'$\Omega_m={Omega_m}$, $\Omega_\Lambda={Omega_l}$')

ax1.set_xlabel('z', fontsize = 14)
ax2.set_xlabel('z', fontsize = 14)

ax1.set_ylabel('$D_A \, / \, D_H$', fontsize = 14)
ax2.set_ylabel('$D_L \, / \, D_H$', fontsize = 14)

ax1.set_title('Dimensionless Angular Diameter Distance for Different
   Universes')
ax2.set_title('Dimensionless Luminosity Distance for Different
   Universes')

ax1.legend()
ax2.legend()

# ax1.get_figure().savefig('Angular Distance Plot.png', dpi=300)
# ax2.get_figure().savefig('Luminosity Distance Plot.png', dpi=300)

plt.show()

ang_distances_1, ang_distances_2, ang_distances_3 = ang_distances
lum_distances_1, lum_distances_2, lum_distances_3 = lum_distances
```

```
all_distances = (ang_distances_1, ang_distances_2, ang_distances_3,
   lum_distances_1, lum_distances_2, lum_distances_3)

Z_values = [1, 1.25, 2.0, 4.0]

for l,k in enumerate(all_distances):
    for z_val in Z_values:
        print(f'Distance {l+1} for z = {z_val}')

        for i,j in enumerate(Z):
            if round(j,3) == z_val:
                print(f'{k[i]:.2f}')
        print('\n')
    print('------------------------------')
```

comoving.py - used in Question 6 and 7

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.optimize import brentq

df = pd.read_csv("/Users/ishu/computing/CATAM/Cosmological Distances/
   quasar.csv")

df[['z', 'f/f0']] = df['z      f/f0'].str.split(n=1, expand=True) # .
   dat file had data all together so this separates into actual
   columns

df['z'] = pd.to_numeric(df['z'])  # convert from string values to
   float
df['f/f0'] = pd.to_numeric(df['f/f0'])

df.drop(columns=['z      f/f0'], inplace=True)

z_values = df['z'].values
f_over_f0 = df['f/f0'].values

# Constants
Mpc = 3.0856e22  # m
c = 2.998e8  # m/s

H_0 = 72e3 / Mpc  # s^-1
t_H = 1 / H_0
D_H = c / H_0  # Hubble distance

def E(z):
    return np.sqrt(Omega_m * (1 + z)**3 + Omega_l)

def distance_func(z):
    return D_H * (1 / E(z))

def midpoint_rule(func, a, b, n):
    h = (b - a) / n
    integral = 0
    for i in range(n):
        midpoint = a + (i + 0.5) * h
        integral += func(midpoint)
    integral *= h
    return integral

def distance_integral(z):  # this is D_C
    return midpoint_rule(distance_func, 0, z, 1000)

def volume(z):
    return (4 * np.pi * distance_integral(z)**3) / 3
```

```
V = []
Z = z_values

Omega_m_values = [1, 0.27]
Omega_l_values = [0, 0.73]

for i in range(0, 2):
    Omega_m = Omega_m_values[i]
    Omega_l = Omega_l_values[i]
    V.append([volume(z) for z in Z])

V1,V2 = V
print(V1)
print(V2)

### FINDING Z_MAX
def calc_z_max(D_Lmax):
    def nested_func(z):
        D_C = distance_integral(z)
        return (1 + z) * D_C - D_Lmax #eqn 22 in project but set to 0

    z_max, _ = brentq(nested_func, 0, 10, full_output=True)
    return z_max

Omega_m = 0.27
Omega_l = 0.73
z_maxes = np.zeros_like(z_values)

for i in range(0,114):
    D_L = (1 + z_values[i]) * distance_integral(z_values[i])
    D_Lmax = D_L * np.sqrt(f_over_f0[i])
    z_max = calc_z_max(D_Lmax)
    z_maxes[i] = z_max

V_max2 = volume(z_maxes)
a = V2 / V_max2
print(f'<V/V_max> = {np.mean(a)}')

small_z = []
small_f_f0 = []

for i in range(0,114):
    if z_values[i] < 0.3:
        small_z.append(z_values[i])
        small_f_f0.append(f_over_f0[i])

small_z = np.array(small_z)
small_f_f0 = np.array(small_f_f0)

small_z_maxes = np.zeros_like(small_z)
```

```
for i in range(0,len(small_z)):
    D_L = (1 + small_z[i]) * distance_integral(small_z[i])
    small_D_Lmax = D_L * np.sqrt(small_f_f0[i])
    small_z_max = calc_z_max(small_D_Lmax)
    small_z_maxes[i] = small_z_max


v2 = volume(small_z)
v_max2 = volume(small_z_maxes)

volume_ratio = v2 / v_max2
f_ratio = small_f_f0**(-3/2)

def theoretical_model(x,alpha, beta):
    return (alpha * x) + beta

popt, pcov = curve_fit(theoretical_model, f_ratio, volume_ratio)
alpha_fit = popt[0]
beta_fit = popt[1]

plt.figure(figsize=(7,5))
plt.scatter(f_ratio, volume_ratio, alpha = 0.9, marker = 'x',s=70,
   label = '$V/V_{{max}} \, $ test data')
plt.plot(f_ratio, theoretical_model(f_ratio, alpha_fit, beta_fit),
   color = 'r', label = f'Linear fit, $V/V_{{max}}$ = ${alpha_fit:.4f}
    \, (f/f_0)^{{-3/2}} + {beta_fit:.4f}$')
plt.title('Euclidean limit')
plt.xlabel('$(f/f_0)^{{-\\frac{{3}}{{2}}}}$', fontsize = 12)
plt.ylabel('$\\frac{{V}}{{V_{{max}}}}$', fontsize = 14)
plt.legend(fontsize = 11)
# plt.savefig('Euclidean limit', dpi=300, bbox_inches='tight')
plt.show()

print(alpha_fit)
print(beta_fit)
```