Cosmology Programming Assignment 4
Cosmological Parameters from Type Ia SNe photometric data

Kiran L
SC17B150

24 March 2021

1 Introduction

The luminosity distance is a way of expressing the amount of light received from a distant object. Let us suppose we observe an object with a certain flux. The luminosity distance is the distance that the object appears to have, assuming the inverse square law for the reduction of light intensity with distance holds. The luminosity distance is not the physical distance to the object, because in the real Universe the inverse square law does not hold. It is broken both because the geometry of the Universe need not be flat, and the universe is expanding. For generality, while in the following discussion it is presumed the object is observed at the present epoch.

We begin with definitions as follows. The luminosity L of an object is defined as the energy emitted per unit solid angle per second; since the total solid angle is 4π steradians, this equals the total power output divided by 4π. The radiation flux density S received by us is defined as the energy received per unit area per second. Then

$$d_{lum}^2 = \frac{L}{S} \tag{1}$$

because L/ S is the unit area per unit solid angle.

This is best visualized by placing the radiating object at the centre of a sphere, co-moving radius r_o, with us holding our detector at the surface of the sphere, as shown in Figure 1. The physical radius of the sphere is $a_o r_o$, and so its total surface area is $4\pi a_o^2 r_o^2$. In this representation, the effect of the geometry is in the determination of r_o; it doesn't appear explicitly in the area.

If we were in a static space the radiation flux received would simply be $S = \frac{L}{a_o^2 r_o^2}$, but we have to allow for the expansion of the Universe and how that

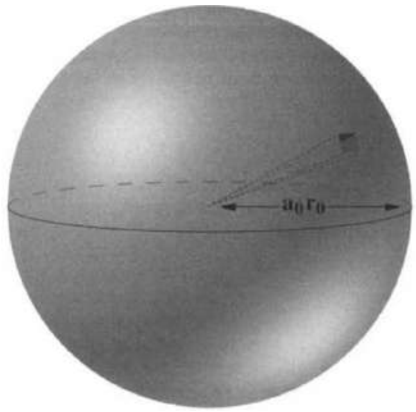

Figure 1: We receive light a distance $a_o r_o$ from the source. The surface area of the sphere at that distance is $4\pi a_o^2 r_o^2$, and so our detector of unit area intercepts a fraction $1/4\pi a_o^2 r_o^2$ of the total light output 4π L.

affects the photons as they propagate from the source to the observer. There arc actually two effects:

- The individual photons lose energy $\propto$ (1 + z), so have less energy when they arrive.
- The photons arrive less frequently $\propto (1+z)$.

Combining the two, the received flux is

$$S = \frac{L}{a_o^2 r_o^2 (1+z)^2} \tag{2}$$

and hence, the luminosity distance is given by

$$d_{lum} = a_o r_o (1+z) \tag{3}$$

Distant objects appear to be further away than they really are because of the effect of redshift reducing their apparent luminosity. For example, consider a flat spatial geometry k = 0. Then for a radial ray ds = a(t)dr and so the physical distance to a source is given by integrating this at fixed time

$$d_{phy} = a_o r_o \tag{4}$$

For nearby objects z « 1 and so $d_{1um} \simeq d_{phys}$. i.e. the objects really are just as far away as they look. But more distant objects appear further away ($d_{1um} > d_{phys}$) than they really are.

Distance Modulus: The difference between the apparent and absolute magnitude is defined as the distance modulus. The relation between the distance modulus and luminosity distance is:

$$m - M = 5log(d(MPc)) + 25 \tag{5}$$

2 Data Visualization

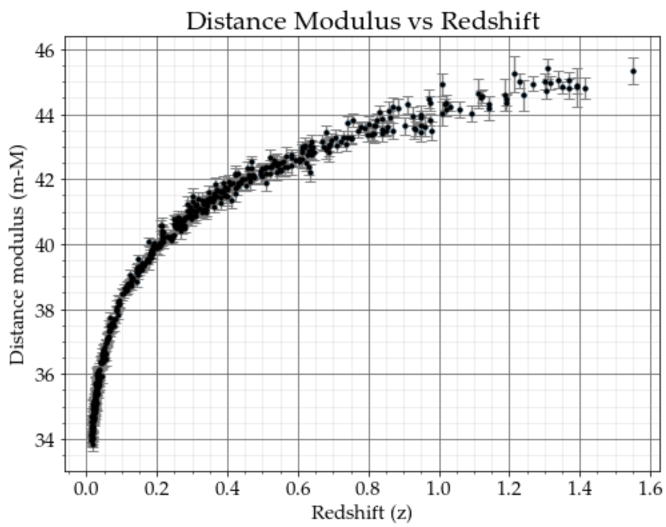

Figure 2: Plot of distance modulus as a function of redshift

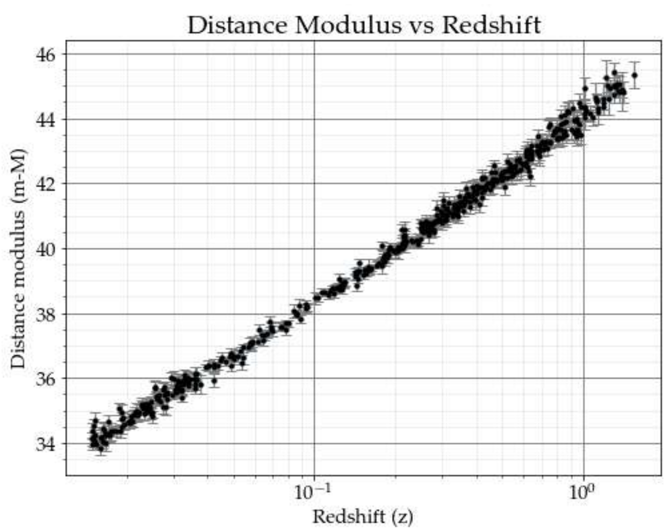

Figure 3: Semi-log plot of distance modulus as a function of redshift

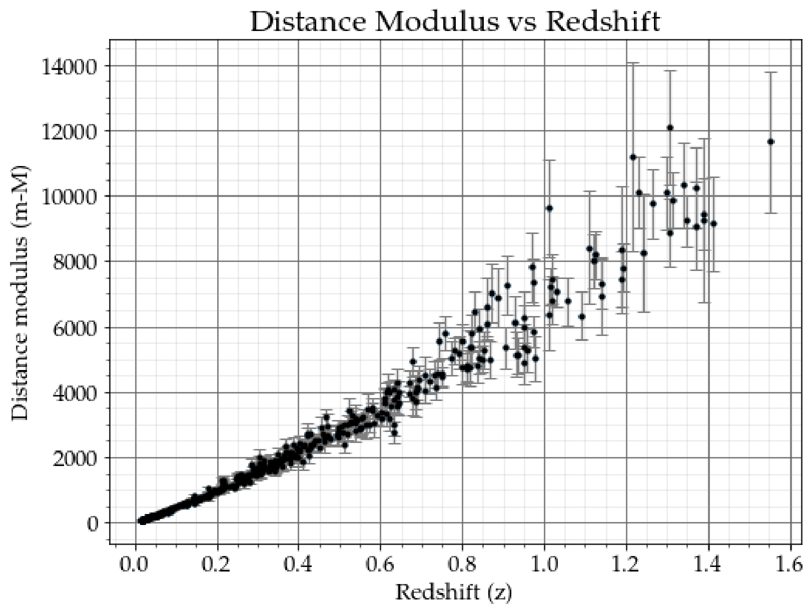

Figure 4: Plot of luminosity distance as a function of redshift

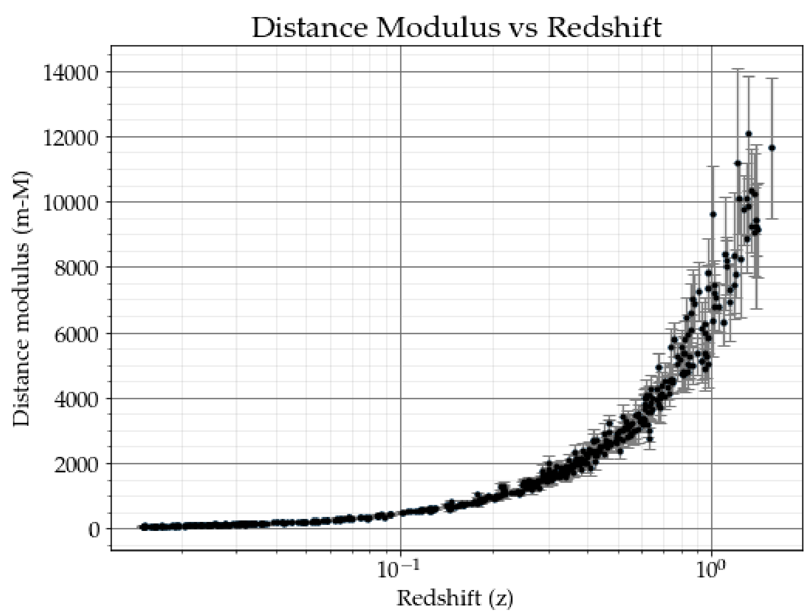

Figure 5: Semi-Log plot of luminosity distance as a function of redshift

3 Results

The data fitting was performed upon the given data of distance modulus as well as by transforming the distance modulus values into luminosity distances. Also, the data fitting was performed using different methods - by simple constrained ($\Omega_m + \Omega_\Lambda$) grid search, constrained as well as un-constrained curve-fitting using scipy module functions. The results obtained are described in the sections below.

The tables below (Table 1 and 2) represent the summary of results obtained upon fitting different models (of flat universe) to the given data of distance modulus.

Method	Ω_m	Ω_Λ	χ_ν	p-value
Constrained Grid Search	0.3	0.7	0.95	1.0
Scipy (constrained) Curvefit	0.3	0.7	0.95	1.0
Scipy (un-constrained) Curvefit	0.29	0.72	0.949	1.0

Table 1: Best fit models determined using different methods by fitting distance modulus data to the models.

Method	Ω_m	Ω_Λ	χ_ν	p-value
Constrained Grid Search	0.33	0.67	0.934	1.0
Scipy (constrained) Curvefit	0.3	0.7	0.943	1.0
Scipy (un-constrained) Curvefit	0.29	0.72	0.938	1.0

Table 2: Best fit models determined using different methods by fitting luminosity distance data to the models.

3.1 Fitting to distance modulus data

3.1.1 Constrained Grid Search Method

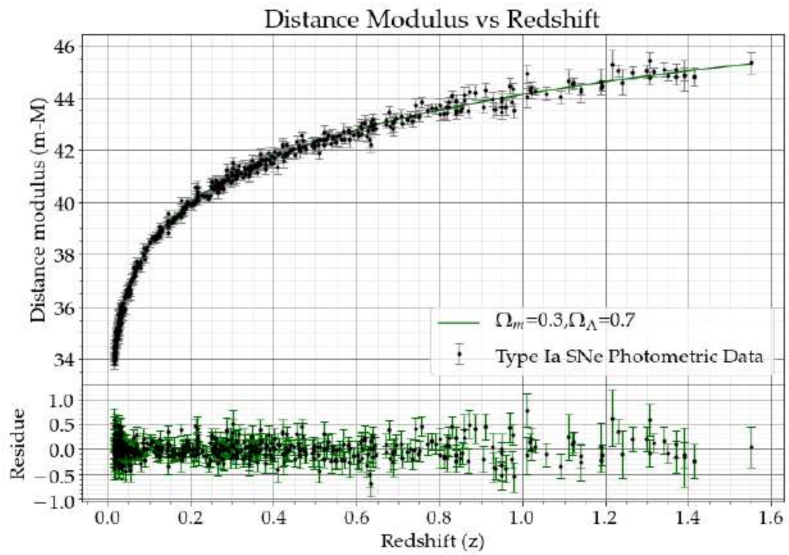

Figure 6: Plot of data of distance modulus as a function of redshift along with the best fit model

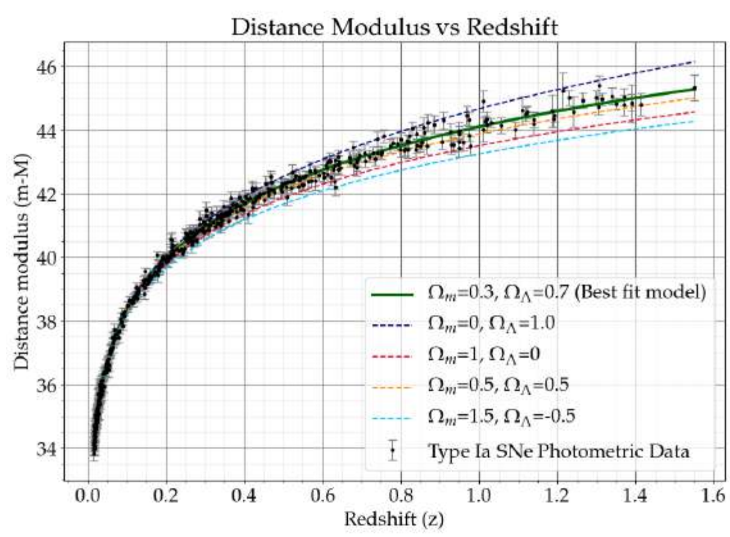

Figure 7: Plot of data of distance modulus as a function of redshift along with the different models

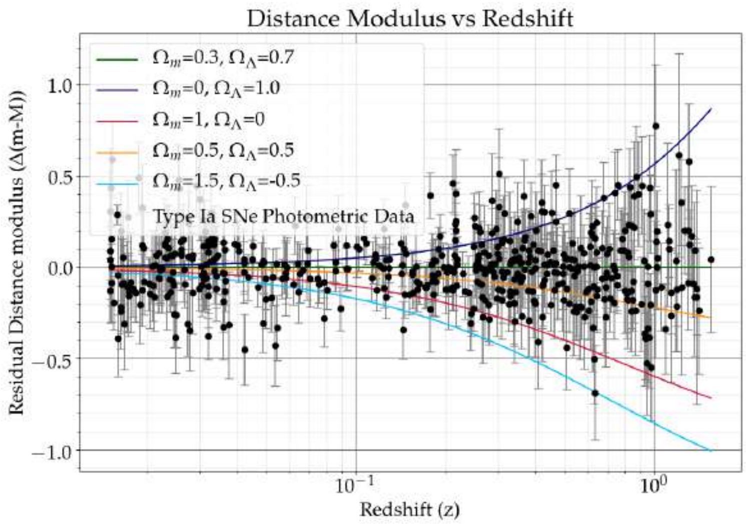

Figure 8: Residual plot of distance modulus as a function of redshift: plot indicates the difference between observed as well as predicted distance modulus (from different models) and that predicted by the best model.

3.1.2 Constrained Curve-fitting Method (using Scipy)

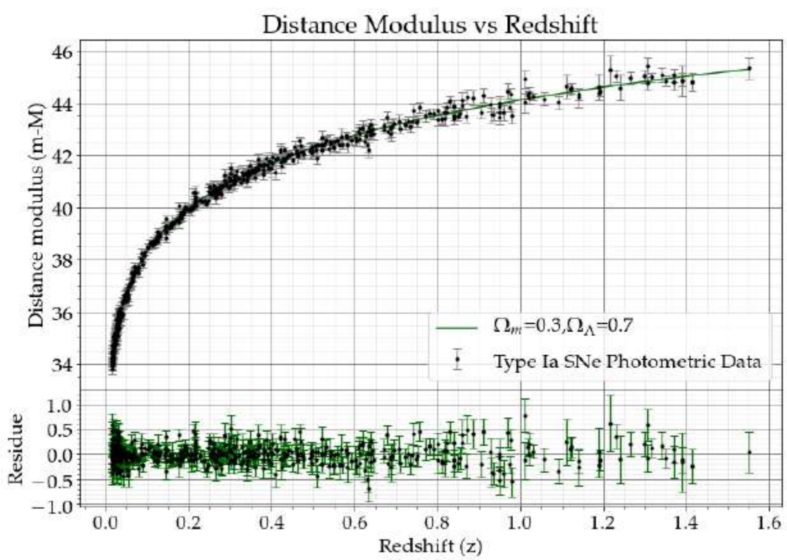

Figure 9: Plot of data of distance modulus as a function of redshift along with the best fit model

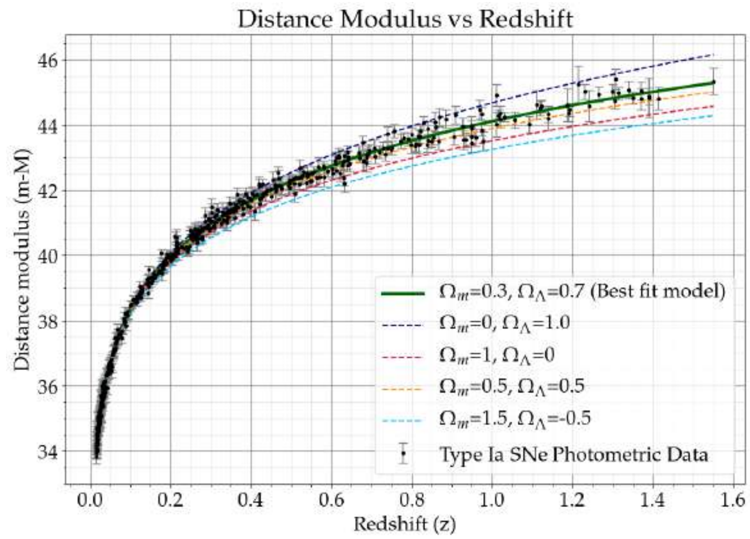

Figure 10: Plot of data of distance modulus as a function of redshift along with the different models

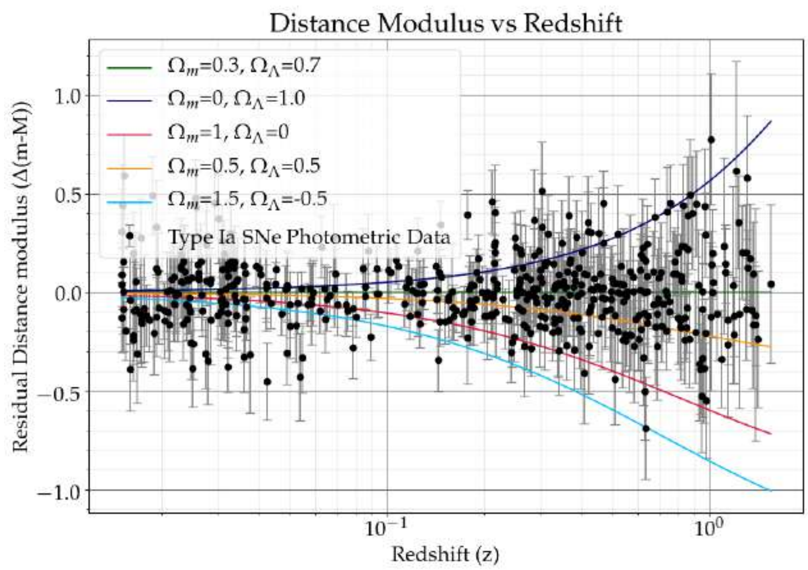

Figure 11: Residual plot of distance modulus as a function of redshift: plot indicates the difference between observed as well as predicted distance modulus (from different models) and that predicted by the best model.

3.1.3 Un-Constrained Curve-fitting Method (using Scipy)

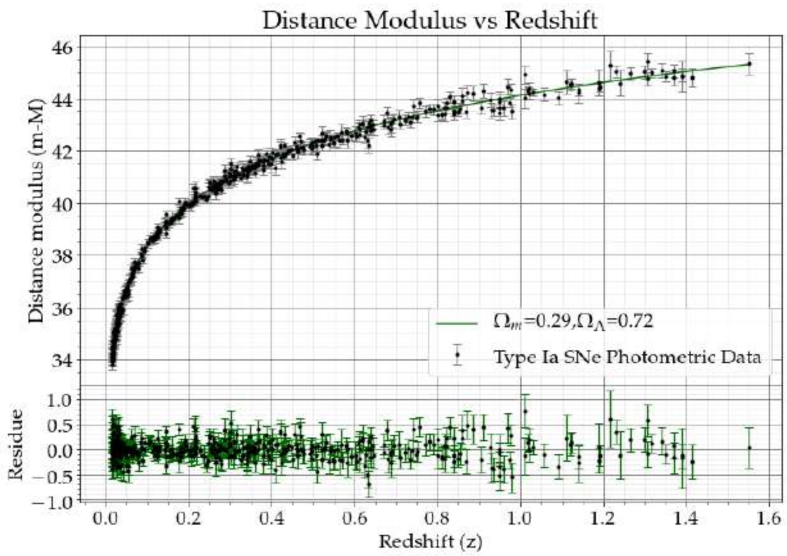

Figure 12: Plot of data of distance modulus as a function of redshift along with the best fit model

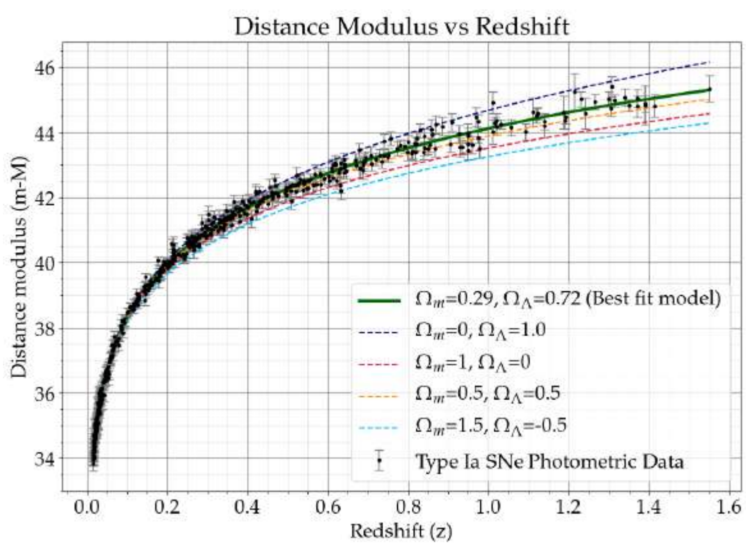

Figure 13: Plot of data of distance modulus as a function of redshift along with the different models

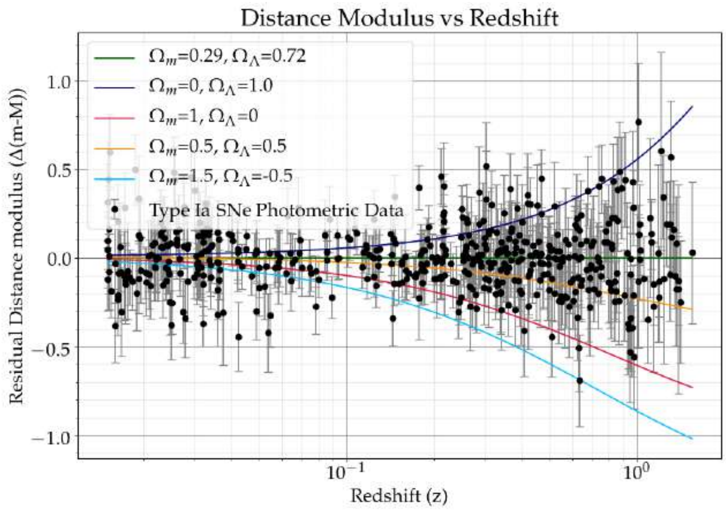

Figure 14: Residual plot of distance modulus as a function of redshift: plot indicates the difference between observed as well as predicted distance modulus (from different models) and that predicted by the best model.

3.2 Fitting to luminosity distance data

3.2.1 Constrained Grid Search Method

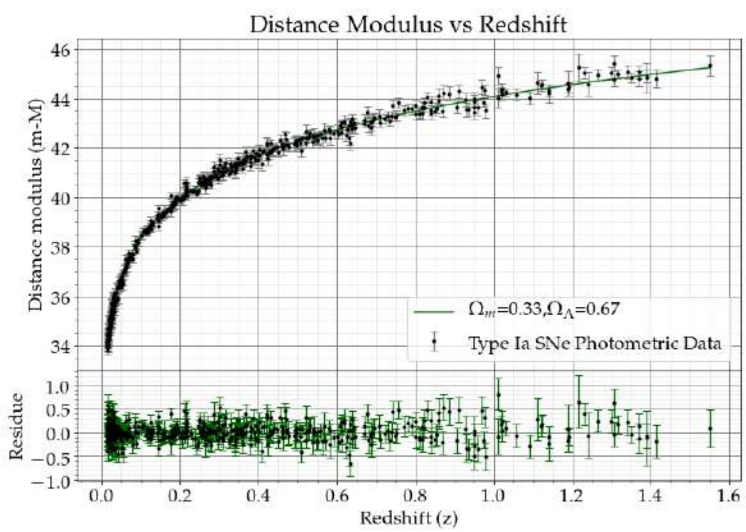

Figure 15: Plot of data of distance modulus as a function of redshift along with the best fit model

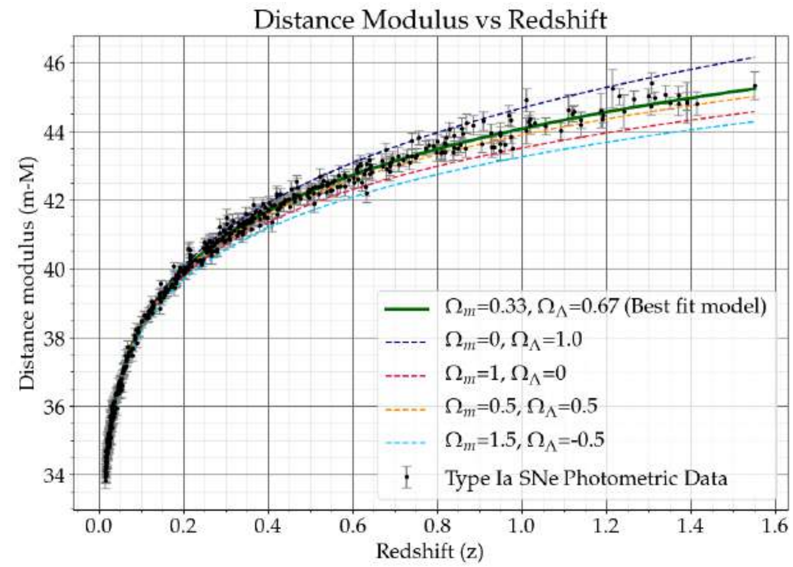

Figure 16: Plot of data of distance modulus as a function of redshift along with the different models

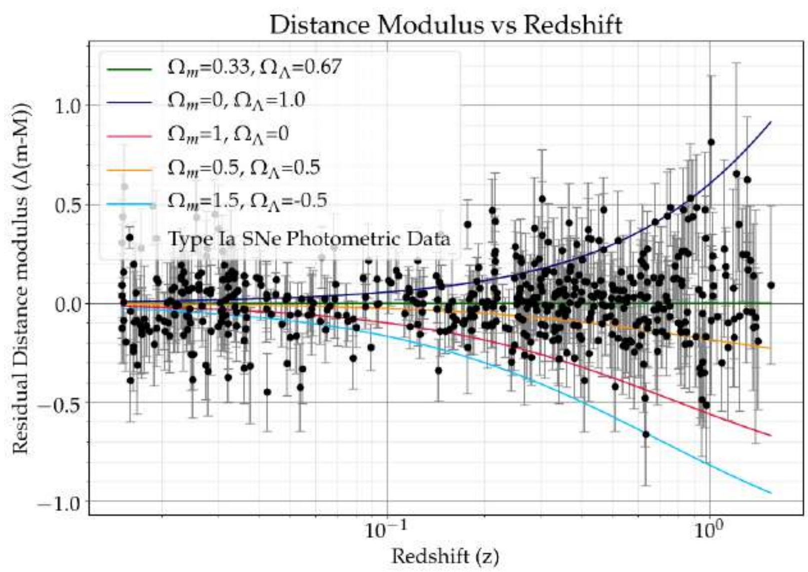

Figure 17: Residual plot of distance modulus as a function of redshift: plot indicates the difference between observed as well as predicted distance modulus (from different models) and that predicted by the best model.

3.2.2 Constrained Curve-fitting Method (using Scipy)

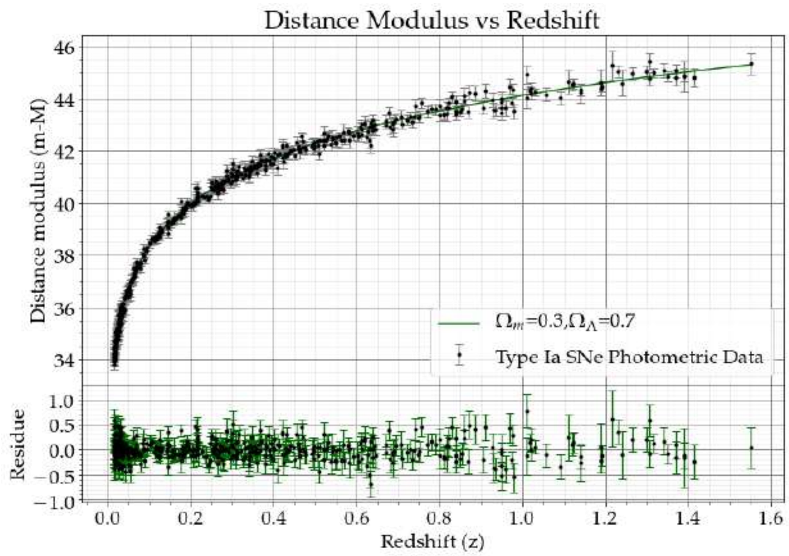

Figure 18: Plot of data of distance modulus as a function of redshift along with the best fit model

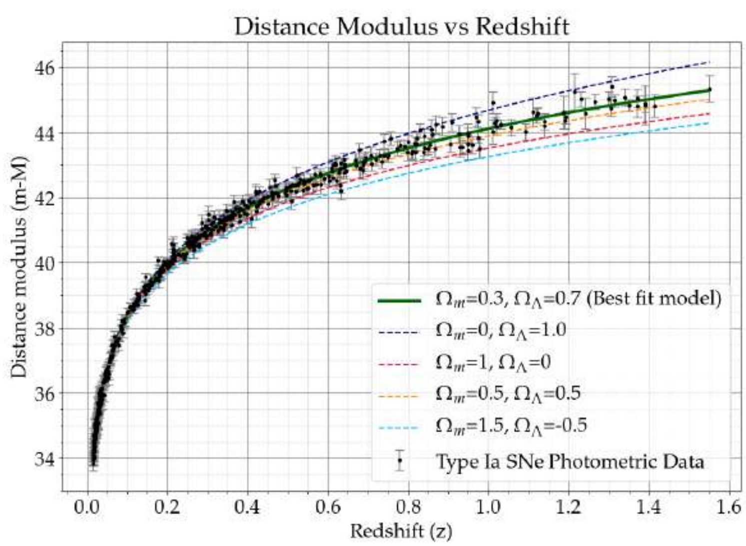

Figure 19: Plot of data of distance modulus as a function of redshift along with the different models

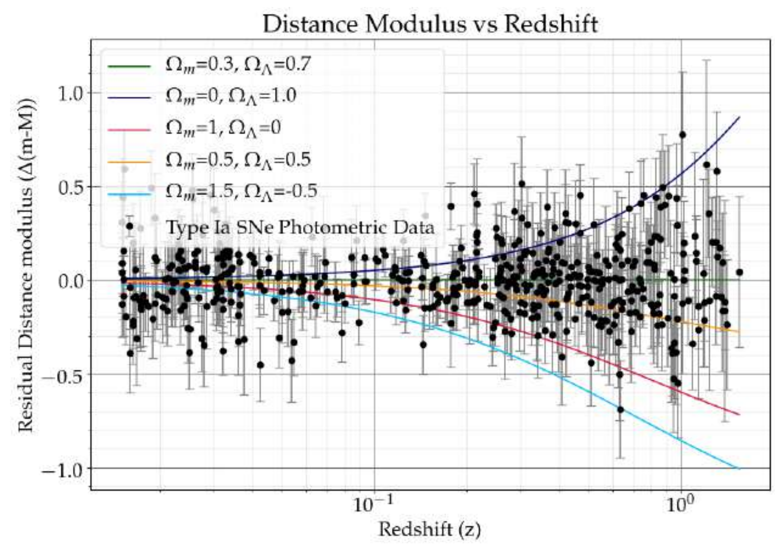

Figure 20: Residual plot of distance modulus as a function of redshift: plot indicates the difference between observed as well as predicted distance modulus (from different models) and that predicted by the best model.

3.2.3 Un-Constrained Curve-fitting Method (using Scipy)

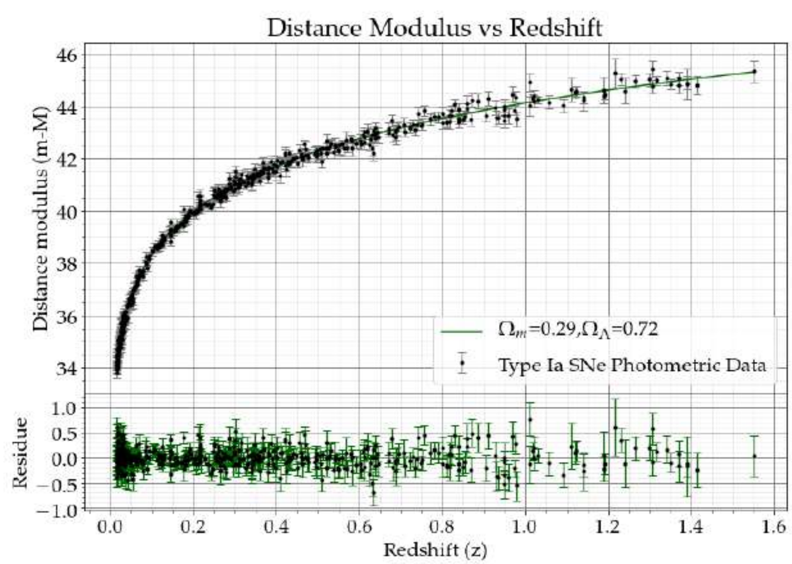

Figure 21: Plot of data of distance modulus as a function of redshift along with the best fit model

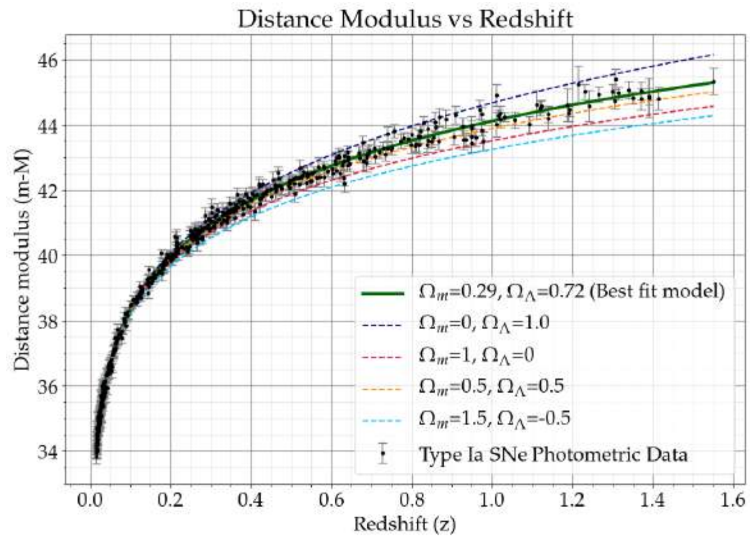

Figure 22: Plot of data of distance modulus as a function of redshift along with the different models

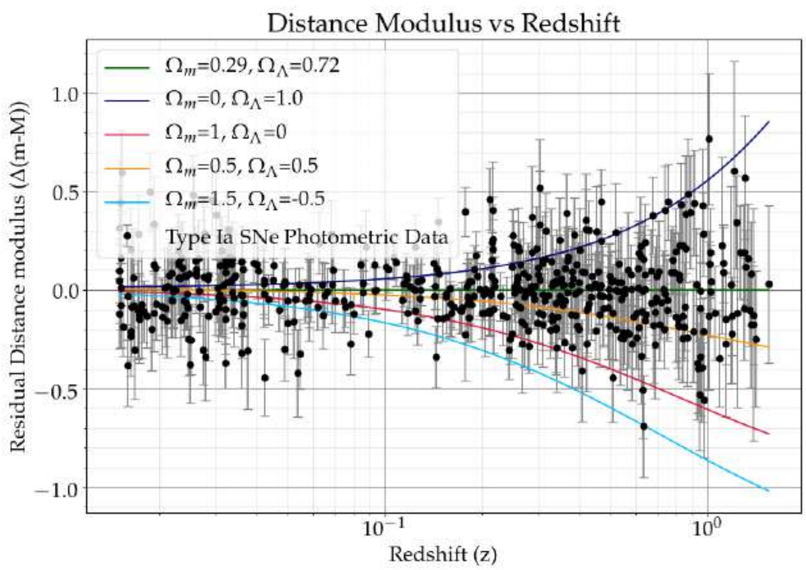

Figure 23: Residual plot of distance modulus as a function of redshift: plot indicates the difference between observed as well as predicted distance modulus (from different models) and that predicted by the best model.

4 Observations and Inferences

Sl. No.	**Observations**	**Inferences**
1.	From Figure 3 we observe that the data is uniformly distributed across the different redshift values considered.	Thus, we have sufficiently well sampled data (sampled along the redshift axis) which aids in obtaining a statistically good fit to the data.
2.	From Tables 1 and 2 we observe that the parameter values obtained by fitting distance modulus data and luminosity data with models by constrained grid search are different.	We can infer this to be the issue due to finite step size of the grid search. This also emphasizes the need for proper data representation while using grid search method, else, one is likely to miss the local minima when not using small enough step size.
3.	From Tables 1 and 2 we observe that the parameter values obtained by fitting distance modulus data and luminosity data with models by constrained and un-constrained curve-fitting methods (using scipy's module functions) are quite similar to each other.	We can infer that the fitting algorithm used by scipy is robust enough to determine the best fit parameter values irrespective of the form of the data.
4.	From the plots of the different model predicted distance modulus (or equivalently, the luminosity distance) we observe that with increase in the dark energy content of the universe leads to increase in the distance modulus value.	We infer that the dark energy contributes to increase in separation between the galaxies thus leading to the galaxies appearing to be dimmer than expected in a matter-dominated universe.

5 Conclusion

Thus, in the present analysis, we have fit the data of distance modulus of Type Ia Supernovae located at different redshifts to best fit model obtained assuming flat universe with negligible contribution of radiation. We have also transformed the distance modulus data to the luminosity distance and fit models to the same. We observe that the results obtained in both these approaches matches when using scipy's curve-fit function, whereas we observe a significant mis-match between the parameter values predicted when using constrained grid search method (under the constraint: $\Omega_m + \Omega_\Lambda = 1$), possibly due to finite size of the grid search step. However, we obtain values of Ω_m and Ω_Λ that are close to the values determined by Supernova Cosmology Project team using the complete dataset (here, we have used only sub-set of the complete dataset - considering only data with low photometric errors).

6 Program

```python
##############################
#### Importing Libraries ####
##############################

import numpy as np
import matplotlib.pyplot as plt
from astropy import units as u
import matplotlib as mpl
from astropy import constants as const
from scipy import integrate
from scipy import stats
from scipy.optimize import curve_fit
import pandas as pd
plt.rcParams.update({
    "text.usetex": True,
    "font.family": "serif",
    "font.serif": ["Palatino"],
})

##############################
#### Function Definition ####
##############################

def visualize(x,y,dy=None,title="data",typ="logx"):
    fs = 15
    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(1,1,1)
    ax.errorbar(x,y,dy,fmt='.',color="k",capsize=4,ecolor="gray",elinewidth
    =1.3)
    ax.scatter(x,y,s=10)
    if(typ=="logx"):
        ax.set_xscale("log")
    plt.xlabel("Redshift (z)",fontsize=fs)
    plt.ylabel(r"Distance modulus (m-M)",fontsize=fs)
    plt.xticks(fontsize=fs)
    plt.yticks(fontsize=fs)
    plt.grid(b=True, which='major', color='#666666', linestyle='-')
    plt.minorticks_on()
    plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha
    =0.2)
    plt.title("Distance Modulus vs Redshift",fontsize=fs+5)
    plt.savefig("{}.png".format(title),bbox_inches="tight")
    plt.close()

def H(z,omega_m,omega_lam):
    return H0*np.sqrt(E(z,omega_m,omega_lam))

def Integrand(z,omega_m,omega_lam):
    return 1.0/H(z,omega_m,omega_lam)

def get_dL(z,omega_m,omega_lam,unit="Mpc"):
    integrand = lambda z: Integrand(z,omega_m=omega_m,omega_lam=omega_lam)
    if(unit == "m"):
        d,dd = integrate.quad(integrand,0,z)
        d *= (c*(1+z))
        dd *= (c*(1+z))
        return [d,dd]
    if(unit == "Mpc"):
        if(isinstance(z, list) or isinstance(z, np.ndarray)):
            d,dd = np.zeros(len(z)),np.zeros(len(z))
            for i in range(len(z)):
                d[i],dd[i] = integrate.quad(integrand,0,z[i])
        else:
            d,dd = integrate.quad(integrand,0,z)
        d *= ((c*m_to_Mpc)*(1+z))
```

```python
        dd *= ((c*m_to_Mpc)*(1+z))
        return [d,dd]

def reduced_chi_square(x,y,s,m): # ddof = v
    v = x.size - m
    chi2 = (np.sum((x-y)**2/s**2))/v
    p = 1 - stats.chi2.cdf(chi2, v)
    return chi2,p

def E(z,omega_m,omega_lam):
    return omega_m*(1+z)**3 + omega_lam

def dm_to_dL(dm,dmerr):
    dL = 10**((dm-25.0)/5.0) # in Mpc
    dLerr = dL*np.log(10)*(dmerr/5.0)
    return dL,dLerr

def dL_to_dm(dL,dLerr):
    dm = 5*np.log10(dL) + 25
    dmerr = (dLerr*5*np.log(10))/dL
    return dm,dmerr

def get_dm_arr(z_arr,omega_m,omega_lam=None):
    if(omega_lam==None):
        dL = np.array([get_dL(z,omega_m,1-omega_m) for z in z_arr])
    else:
        dL = np.array([get_dL(z,omega_m,omega_lam) for z in z_arr])
    dL_arr = dL.T[0]
    ddL_arr = dL.T[1]
    return dL_to_dm(dL_arr,ddL_arr)

def fit_func_unconstrained(z,omega_m,omega_lam):
    dL = get_dL(z,omega_m,omega_lam,unit="Mpc")
    dm,dm_err = dL_to_dm(dL[0],dL[1])
    return dm

def fit_func_constrained(z,omega_m):
    dL = get_dL(z,omega_m,1-omega_m,unit="Mpc")
    dm,dm_err = dL_to_dm(dL[0],dL[1])
    return dm

def fit_func_unconstrained_L(z,omega_m,omega_lam):
    dL = get_dL(z,omega_m,omega_lam,unit="Mpc")
    return dL[0]

def fit_func_constrained_L(z,omega_m):
    dL = get_dL(z,omega_m,1-omega_m,unit="Mpc")
    return dL[0]

def plot_best_fit(m,lam,method="grid_search",typ="normal"):
    fs=20.0
    fig1 = plt.figure(1,figsize=(12,8))
    #Plot Data-model
    frame1 = fig1.add_axes((.1,.3,.8,.6))
    plt.errorbar(data["z"],data["dm"],yerr=data["dmerr"],fmt='.',color="black",capsize=4,ecolor='gray',elinewidth=1.3, label="Type Ia SNe Photometric Data")
    if(typ=="errorbar"):
        plt.errorbar(x=z_arr,y=dm_arr["best"],yerr=ddm_arr["best"],color="darkgreen",capsize=4,ecolor='darkgreen',elinewidth=1.3,label=r"$\Omega_m$={},$\Omega_\Lambda$={}".format(np.round(m,2),np.round(lam,2)))
    else:
        plt.plot(z_arr,dm_arr["best"],color="darkgreen",label=r"$\Omega_m$={},$\Omega_\Lambda$={}".format(np.round(m,2),np.round(lam,2)))
    frame1.set_xticklabels([]) #Remove x-tic labels for the first frame
    plt.xlabel("Redshift (z)",fontsize=fs)
    plt.ylabel(r"Distance modulus (m-M)",fontsize=fs)
    plt.xticks(fontsize=fs)
```

```python
    plt.yticks(fontsize=fs)
    plt.grid(b=True, which='major', color='#666666', linestyle='-')
    plt.minorticks_on()
    plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
    plt.title("Distance Modulus vs Redshift",fontsize=fs+5)
    plt.legend(fontsize=fs)
    #Residual plot
    difference = data["dm"] - dm["best"]
    frame2=fig1.add_axes((.1,.1,.8,.2))
    plt.errorbar(data["z"],difference,yerr=ddm["best"] + data["dmerr"],fmt='.',color="black",capsize=4,ecolor='darkgreen',elinewidth=1.3)
    plt.ylabel("Residue",fontsize=fs)
    plt.xlabel("Redshift (z)",fontsize=fs)
    plt.xticks(fontsize=fs)
    plt.yticks(fontsize=fs)
    plt.grid(b=True, which='major', color='#666666', linestyle='-')
    plt.minorticks_on()
    plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
    plt.savefig("best_fit_with_residual_plot_{}_{}.png".format(method,typ), bbox_inches="tight")
    plt.close()

def plot_diff_models(m,lam,method="grid_search",typ="normal",scale="logx"):
    fs=20.0
    fig1 = plt.figure(1,figsize=(12,8))
    plt.errorbar(data["z"],data["dm"],yerr=data["dmerr"],fmt='.',color="black",capsize=4,ecolor='gray',elinewidth=1.3,label="Type Ia SNe Photometric Data")
    if(typ == "errorbar"):
        plt.errorbar(x=z_arr,y=dm_arr["best"],yerr=ddm_arr["best"],color="darkgreen",capsize=4,ecolor='darkgreen',elinewidth=1.3,label=r"$\Omega_m$={}, $\Omega_\Lambda$={} (Best fit model)".format(np.round(m,2), np.round(lam,2) ))
        plt.errorbar(x=z_arr,y=dm_arr["0"],yerr=ddm_arr["0"],color="navy",capsize=4,ecolor='navy',elinewidth=1.3,label=r"$\Omega_m$={}, $\Omega_\Lambda$={}".format(0,1.0))
        plt.errorbar(x=z_arr,y=dm_arr["1"],yerr=ddm_arr["1"],color="crimson",capsize=4,ecolor='crimson',elinewidth=1.3,label=r"$\Omega_m$={}, $\Omega_\Lambda$={}".format(1,0))
        plt.errorbar(x=z_arr,y=dm_arr["0.5"],yerr=ddm_arr["0.5"],color="darkorange",capsize=4,ecolor='darkorange',elinewidth=1.3,label=r"$\Omega_m$={}, $\Omega_\Lambda$={}".format(0.5,0.5))
        plt.errorbar(x=z_arr,y=dm_arr["1.5"],yerr=ddm_arr["1.5"],color="magenta",capsize=4,ecolor='magenta',elinewidth=1.3,label=r"$\Omega_m$={}, $\Omega_\Lambda$={}".format(1.5,-0.5))
    else:
        plt.plot(z_arr,dm_arr["best"],lw=3,color="darkgreen",label=r"$\Omega_m$={}, $\Omega_\Lambda$={} (Best fit model)".format(np.round(m,2), np.round(lam,2) ))
        plt.plot(z_arr,dm_arr["0"],'--',color="navy",label=r"$\Omega_m$={}, $\Omega_\Lambda$={}".format(0,1.0))
        plt.plot(z_arr,dm_arr["1"],'--',color="crimson",label=r"$\Omega_m$={}, $\Omega_\Lambda$={}".format(1,0))
        plt.plot(z_arr,dm_arr["0.5"],'--',color="darkorange",label=r"$\Omega_m$={}, $\Omega_\Lambda$={}".format(0.5,0.5))
        plt.plot(z_arr,dm_arr["1.5"],'--',color="magenta",label=r"$\Omega_m$={}, $\Omega_\Lambda$={}".format(1.5,-0.5))
    if(scale=="logx"):
        plt.xscale("log")
    plt.xlabel("Redshift (z)",fontsize=fs)
    plt.ylabel(r"Distance modulus (m-M)",fontsize=fs)
    plt.xticks(fontsize=fs)
    plt.yticks(fontsize=fs)
    plt.grid(b=True, which='major', color='#666666', linestyle='-')
    plt.minorticks_on()
```

```
    plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha
     =0.2)
    plt.title("Distance Modulus vs Redshift",fontsize=fs+5)
    plt.legend(fontsize=fs,loc="lower right")
    plt.savefig("different_models_{}_{}_{}.png".format(method,typ,scale),
     bbox_inches="tight")
    plt.close()

def plot_residuals(m,lam,method="grid_search",typ="normal",scale="logx"):
    fs=20.0
    fig1 = plt.figure(1,figsize=(12,8))
    plt.errorbar(data["z"],data["dm"]-dm["best"],yerr=ddm["best"]+data["dmerr
     "],fmt='o',color="black",capsize=4,ecolor='gray',elinewidth=1.3,label="
     Type Ia SNe Photometric Data")
    if(typ=="errorbar"):
        plt.errorbar(x=data["z"],y=dm["best"]-dm["best"],yerr=ddm["0"]+ddm["
     best"],color="darkgreen",capsize=2,ecolor='darkgreen',elinewidth=1.3,
     label=r"$\Omega_m$={}, $\Omega_\Lambda$={}".format(np.round(m,2),np.
     round(lam,2)))
        plt.errorbar(x=data["z"],y=dm["0"]-dm["best"],yerr=ddm["0"]+ddm["best
     "],color="navy",capsize=2,ecolor='navy',elinewidth=1.3,label=r"$\
     Omega_m$={}, $\Omega_\Lambda$={}".format(0,1.0))
        plt.errorbar(x=data["z"],y=dm["1"]-dm["best"],yerr=ddm["1"]+ddm["best
     "],color="crimson",capsize=2,ecolor='crimson',elinewidth=1.3,label=r"$\
     Omega_m$={}, $\Omega_\Lambda$={}".format(1,0))
        plt.errorbar(x=data["z"],y=dm["0.5"]-dm["best"],yerr=ddm["0.5"]+ddm["
     best"],color="darkorange",capsize=2,ecolor='darkorange',elinewidth=1.3,
     label=r"$\Omega_m$={}, $\Omega_\Lambda$={}".format(0.5,0.5))
        plt.errorbar(x=data["z"],y=dm["1.5"]-dm["best"],yerr=ddm["1.5"]+ddm["
     best"],color="magenta",capsize=2,ecolor='magenta',elinewidth=1.3,label=r
     "$\Omega_m$={}, $\Omega_\Lambda$={}".format(1.5,-0.5))
    else:
        plt.plot(z_arr,dm_arr["best"]-dm_arr["best"],color="darkgreen",label=
     r"$\Omega_m$={}, $\Omega_\Lambda$={}".format(np.round(m,2),np.round(lam
     ,2)))
        plt.plot(z_arr,dm_arr["0"]-dm_arr["best"],color="navy",label=r"$\
     Omega_m$={}, $\Omega_\Lambda$={}".format(0,1.0))
        plt.plot(z_arr,dm_arr["1"]-dm_arr["best"],color="crimson",label=r"$\
     Omega_m$={}, $\Omega_\Lambda$={}".format(1,0))
        plt.plot(z_arr,dm_arr["0.5"]-dm_arr["best"],color="darkorange",label=
     r"$\Omega_m$={}, $\Omega_\Lambda$={}".format(0.5,0.5))
        plt.plot(z_arr,dm_arr["1.5"]-dm_arr["best"],color="magenta",label=r"$
     \Omega_m$={}, $\Omega_\Lambda$={}".format(1.5,-0.5))
    if(scale=="logx"):
        plt.xscale("log")
    plt.xlabel("Redshift (z)",fontsize=fs)
    plt.ylabel(r"Residual Distance modulus ($\Delta$(m-M))",fontsize=fs)
    plt.xticks(fontsize=fs)
    plt.yticks(fontsize=fs)
    plt.grid(b=True, which='major', color='#666666', linestyle='-')
    plt.minorticks_on()
    plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha
     =0.2)
    plt.title("Distance Modulus vs Redshift",fontsize=fs+5)
    plt.legend(fontsize=fs,loc="best")
    plt.savefig("different_models_residuals_{}_{}_{}.png".format(method,typ,
     scale),bbox_inches="tight")
    plt.close()

############################
####### Loading Data ########
############################

data = []
with open("SCPUnion2.1_mu_vs_z low error.txt") as f:
    for line in f:
        data.append(line.split()[:3])
data = np.array([[float(i) for i in row ] for row in data[1:]])
```

```python
data = {"z":data.T[0],"dm":data.T[1],"dmerr":data.T[2]}
data["dL"],data["dLerr"] = dm_to_dL(data["dm"],data["dmerr"])

#############################
#### Data Visualization #####
#############################

visualize(data["z"],data["dm"],data["dmerr"],"data_logx")
visualize(data["z"],data["dm"],data["dmerr"],"data","normal")
visualize(data["z"],data["dL"],data["dLerr"],"data_L_logx")
visualize(data["z"],data["dL"],data["dLerr"],"data_L","normal")

#############################
######## Calculation ########
#############################

###############################################################
###--- Method-1: Fitting models to Distance Modulus Data ---###
###############################################################

print("Fitting models to distance modulus data.")
# grid search under the constrain omega_m + omega_lam = 1
c = const.c.value
m_to_pc = 1.0/const.pc.value
m_to_Mpc = (1e-6)/const.pc.value
pc_to_m = const.pc.value
H0 = 69.7 # km/s/Mpc
H0 = H0*(1e3/(1e6*pc_to_m))
omega_m = np.arange(0.2,0.4,0.01)
omega_lambda = 1-omega_m
z_arr = np.linspace(np.amin(data["z"]),np.amax(data["z"]),1000)
rchi2_arr = np.zeros(len(omega_m))
p_arr = np.zeros(len(omega_m))
dL_arr = np.zeros(len(z_arr))
ddL_arr = np.zeros(len(z_arr))
for ctr,(i,j) in enumerate(zip(omega_m,omega_lambda)):
    dL = np.array([get_dL(z,i,j) for z in data["z"]])
    dm,dmerr = dL_to_dm(dL.T[0],dL.T[1])
    rchi2,p = reduced_chi_square(dm,data["dm"],data["dmerr"],1)
    rchi2_arr[ctr] = rchi2
    p_arr[ctr] = p

idx = np.argmin(rchi2_arr)
print("Best fit parameter values determined by constrained grid search: \n omega_m = {}, omega_lam = {}".format(np.round(omega_m[idx],2),np.round(1-omega_m[idx],2)))

popt_unc, pcov_unc = curve_fit(fit_func_unconstrained, data["z"], data["dm"], sigma=data["dmerr"],bounds=[0,1])
print("Best fit parameter values determined by scipy's curve-fit function, without any constraint on the values of omega_m and omega_lam: \n omega_m = {}, omega_lam = {}".format(np.round(popt_unc[0],2),np.round(popt_unc[1],2)))

popt_c, pcov_c = curve_fit(fit_func_constrained, data["z"], data["dm"], sigma=data["dmerr"],bounds=[0,1])
print("Best fit parameter values determined by scipy's curve-fit function, under the constraint that omega_m + omega_lam = 1: \n omega_m = {}, omega_lam = {}".format(np.round(popt_c[0],2),np.round(1-popt_c[0],2)))

best_m = [omega_m[idx],popt_c[0],popt_unc[0]]
best_lam = [1-omega_m[idx],1-popt_c[0],popt_unc[1]]

###############################################################
###---  Method-1: Fitting models to Luminosity Distance  ---###
###############################################################
print("Fitting models to luminosity distance data.")
# grid search under the constrain omega_m + omega_lam = 1
```

```
# fitting dL
rchi2_arr_L = np.zeros(len(omega_m))
p_arr_L = np.zeros(len(omega_m))
dL_arr = np.zeros(len(z_arr))
ddL_arr = np.zeros(len(z_arr))

for ctr,(i,j) in enumerate(zip(omega_m,omega_lambda)):
    dL = np.array([get_dL(z,i,j) for z in data["z"]])
    rchi2,p = reduced_chi_square(dL.T[0],data["dL"],data["dLerr"],1)
    rchi2_arr_L[ctr] = rchi2
    p_arr_L[ctr] = p

idx = np.argmin(rchi2_arr_L)
print("Best fit parameter values determined by constrained grid search: \n
    omega_m = {}, omega_lam = {}".format(np.round(omega_m[idx],2),np.round(1
    -omega_m[idx],2)))

popt_unc_L, pcov_unc_L = curve_fit(fit_func_unconstrained_L, data["z"], data[
    "dm"], sigma=data["dmerr"],bounds=[0,1])
print("Best fit parameter values determined by scipy's curve-fit function,
    without any constraint on the values of omega_m and omega_lam: \n
    omega_m = {}, omega_lam = {}".format(np.round(popt_unc_L[0],2),np.round(
    popt_unc_L[1],2)))

popt_c_L, pcov_c_L = curve_fit(fit_func_constrained_L, data["z"], data["dm"],
     sigma=data["dmerr"],bounds=[0,1])
print("Best fit parameter values determined by scipy's curve-fit function,
    under the constraint that omega_m + omega_lam = 1: \n omega_m = {},
    omega_lam = {}".format(np.round(popt_c_L[0],2),np.round(1-popt_c_L[0],2)
    ))

best_m_L = [omega_m[idx],popt_c[0],popt_unc[0]]
best_lam_L = [1-omega_m[idx],1-popt_c[0],popt_unc[1]]

#############################
########## Plotting #########
#############################

dm_arr = {}
ddm_arr = {}
dm = {}
ddm = {}

dm_arr["0"],ddm_arr["0"] = get_dm_arr(z_arr,0)
dm_arr["1"],ddm_arr["1"] = get_dm_arr(z_arr,1)
dm_arr["0.5"],ddm_arr["0.5"] = get_dm_arr(z_arr,0.5)
dm_arr["1.5"],ddm_arr["1.5"] = get_dm_arr(z_arr,1.5)

dm["0"],ddm["0"] = get_dm_arr(data["z"],0)
dm["1"],ddm["1"] = get_dm_arr(data["z"],1)
dm["0.5"],ddm["0.5"] = get_dm_arr(data["z"],0.5)
dm["1.5"],ddm["1.5"] = get_dm_arr(data["z"],1.5)

methods = ["grid_search","constrained_curvefit","unconstrained_curvefit"]
for method,m,lam in zip(methods,best_m,best_lam):
    dm_arr["best"],ddm_arr["best"] = get_dm_arr(z_arr,m,lam)
    dm["best"],ddm["best"] = get_dm_arr(data["z"],m,lam)
    plot_best_fit(m,lam,method,"normal")
    plot_best_fit(m,lam,method,"errorbar")
    plot_diff_models(m,lam,method,"normal","normal")
    plot_diff_models(m,lam,method,"errorbar","normal")
    plot_diff_models(m,lam,method,"normal","logx")
    plot_diff_models(m,lam,method,"errorbar","logx")
    plot_residuals(m,lam,method,typ="normal",scale="logx")
    plot_residuals(m,lam,method,typ="normal",scale="normal")
    plot_residuals(m,lam,method,typ="errorbar",scale="logx")
    plot_residuals(m,lam,method,typ="errorbar",scale="normal")

```

```
for method,m,lam in zip(methods,best_m_L,best_lam_L):
    dm_arr["best"],ddm_arr["best"] = get_dm_arr(z_arr,m,lam)
    dm["best"],ddm["best"] = get_dm_arr(data["z"],m,lam)
    plot_best_fit(m,lam,"L_" + method,"normal")
    plot_best_fit(m,lam,"L_" + method,"errorbar")
    plot_diff_models(m,lam,"L_" + method,"normal","normal")
    plot_diff_models(m,lam,"L_" + method,"errorbar","normal")
    plot_diff_models(m,lam,"L_" + method,"normal","logx")
    plot_diff_models(m,lam,"L_" + method,"errorbar","logx")
    plot_residuals(m,lam,"L_" + method,typ="normal",scale="logx")
    plot_residuals(m,lam,"L_" + method,typ="normal",scale="normal")
    plot_residuals(m,lam,"L_" + method,typ="errorbar",scale="logx")
    plot_residuals(m,lam,"L_" + method,typ="errorbar",scale="normal")

#############################
######## End Of Code ########
#############################
```

References

[1] *Andrew R. Liddle, An Introduction to Modern Cosmology*