D.S. KOTHARI CENTER FOR RESEARCH AND INNOVATION

DEPARTMENT OF PHYSICS

MIRANDA HOUSE, UNIVERSITY OF DELHI

DSKC Summer Workshop 2022

Area 4: Cosmology, Data Simulation and Parameter Fitting

CONSTRAINING ΛCDM MODEL AND DISPERSION MEASURE OF HOST GALAXIES USING STATISTICAL TOOLS ON FRBS

Submitted by:

Aditi Sharma*

Nupur Rajesh Deshpande**

Rashmi Sarwal*

Shakshi*

Mentors:

Dr. Abha Dev Habib*

Dr. Nisha Rani*

*Miranda House, University of Delhi, India-110007

**IISER Tirupati, Andhra Pradesh, India-517507

DECLARATION

We certify that

a. The work contained in this project has been done by us under the guidance of our supervisors.

b. The work has not been submitted to any other Institute for any degree or diploma.

c. We have followed the guidelines provided by the Institute in preparing the project report.

d. We have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

e. Whenever we have used materials (data, theoretical information, figures, and text) from other sources, we have given due credit to them by citing them in the text of the report and giving their details in the references.

ACKNOWLEDGEMENT

We express immense gratitude towards Miranda House College for giving opportunity for young students in science to come together in the DSKC Summer Workshop and engage in projects. We also thank DSKC for providing us with a workspace and accommodation facilities to work together efficiently as a group.

A major part of our gratitude goes to Dr. Abha Dev Habib and Dr. Nisha Rani for guiding us throughout the duration of the project with their valuable insights and important criticisms. We also thank them for organizing and teaching the lecture series on various statistical tools.

We also thank Dr. Akshay Rana for giving the guest lecture on the Markov Chain Monte Carlo method which forms a major part of our study of the research paper.

Finally, we thank all the students of the DSKC Cosmology group for collectively creating a fun and enthusiastic environment for learning, working, and bonding.

We had a profound experience of learning and developing valuable skills this Summer. We are thankful to everyone involved in making this happen.

INDEX

CHAPTER-1: INTRODUCTION

Cosmology is the study of the physical universe as a whole from the past, present and on to the future. Research in cosmology not only involves astronomy, but also gravitational physics, particle physics, and challenging questions about the interpretation of phenomena we can't see directly — such as the possible existence of something before the Big Bang. Here, we discuss the questions "What is observable?", "What in the Universe is knowable?" and "What are the fundamental limits to cosmological knowledge?" and then describe the methodology for investigation: theoretical hypotheses are used to model, predict and anticipate results; data is used to infer theory.

A cosmological model is a mathematical description of the universe, which tries to explain the reasons of its current aspect, and to describe its evolution during time and it must account for the observations, and be able to make predictions that later observations will be able to check. For this we need a statistical test that makes a quantitative decisions about the prediction made. The goal is to see if there are enough evidences to 'reject' a conjecture or hypothesis. There are many different types of tests for statistics such as t-test, z-test, chi-square test, binomial test, etc. These tests fall into two categories: Parametric Statistics and Non-Parametric Statistics. Parametric statistics are based on assumptions about the distribution of a sampled population. This includes t-tests, z-tests and so. Nonparametric statistics are not based on assumptions, that is, the data can be collected from a sample that does not follow a specific distribution, eg. Chi-sqaure test.

Cosmology has come a long way from being based entirely on a small number of observations to becoming an exact science driven by data. Cosmologists rely on data that contains a lot of useful information. To analyze the data statistical techniques are being employed in cosmology. In cosmology there are 2 distinct aspects of data analysis: model comparison and parameter estimation. In model comparison we try to verify that one model is better than another. In parameter estimation, we've one assumed model and that we are estimating the parameters of that model. We tend to focus here on parameter estimation.

During the period of our project, we learnt various techniques such as the Monte Carlo method, Likelihood analysis, Chi-Square analysis, and Markov Chain Monte Carlo method and practiced these algorithms in a Python environment. We then attempted to replicate the results of a study by using these techniques to generate data. In this study we try to generate data for Fast Radio Bursts for dispersion measure of host galaxies. It also constrains the ΛCDM Model of host galaxies using various statistical tools. We test the model first with 50 data points and then 500 data points. We further aim to verify the results of an interesting paper by Yang Pei Yuan and Bing Zhang1 with observed data and carry out other extended research work. Our main aim is to find the set of parameters that produces the model that best fits our data.

The motivation to choose this paper was that firstly, this paper was based on the applications of various statistical methods we had learned during the period of this summer workshop and also this paper was completely based on computational analysis which we want to explore more. Secondly, this paper was about generating FRBs data and as FRBs are one of the mysterious transients of the universe we were very excited to learn more about it and research undergoing in it.

CHAPTER-II :- STATISTICAL METHODS

Statistical Methods are an essential tool in scientific research. The study of statistical methods involves planning, designing, collecting data, analyzing data and thus developing meaningful interpretations of the data. The results are precise only if an appropriate selection of the study sample and statistical tests are used. For this proper knowledge of statistics is required.
To achieve this, some basic concepts of statistical data analysis such as random variables and probability distributions are introduced with some examples and discussed some of the statistical tools used to infer results from the data.

2.1 Random Variables

Random Variable (Stochastic variable) is a mathematical quantity that depends on random events. That is, the values of the random variable correspond to the outcome of the random experiment. It is a function of possible events in a sample space to a measurable space. Random variables make our task much easier to quantify the results of any random process and apply math and perform further computation.

Random Variables are of two types:

1. Discrete Random Variables
2. Continuous Random Variables

If a countable number of distinct values can be taken by the variable then it is a **discrete random variable.** E.g. tossing a coin.
But if the variable can take an infinite number of values in an interval then it is a **continuous random variable.** E.g. length of rod measured in meters.

2.2 Probability Distribution

It tells us how likely it is that a random variable takes one of its possible states. Thus, mathematically it gives the probability of different outcomes of an experiment.

A probability distribution can be described in two ways:

1. Probability Mass Function
2. Probability Density Function

Probability Mass Function describes the probability distribution of a discrete random variable. Sometimes also known as the discrete density function.
It is the function P(x) = P(X=x) where $-\infty < x < \infty$
It lies in the range of [0,1]. The Sum of all probabilities for every possible state equals one.

Probability Density Function describes the probability distribution of a continuous random variable.

For continuous random variables, we calculate probability over an interval. The Probability Density Function shows where observations are more likely to occur in the probability distribution. Here, the probability of any outcome is zero.

We can use PDFs to calculate the probability by looking at the area under the curve for our interval. This is why the probability of a random variable is 0, as the area of the point is 0.

$$\text{Probability} = P(a \leq x \leq b) = \int_{a}^{b} f(x)dx \geq 0$$

The two out of many types of continuous distribution which are frequently used are: Uniform distribution and Normal distribution (Gaussian distribution).

a. Uniform distribution:

It is a type of probability distribution in which all the outcomes are equally likely to occur. The probability density function (PDF) of a continuous uniform distribution between a and b is as follows:

$$f(x) = \frac{1}{(b-a)}$$

Fig. 1 is the Standard Uniform Distribution plot by generating random numbers between [0,1] .

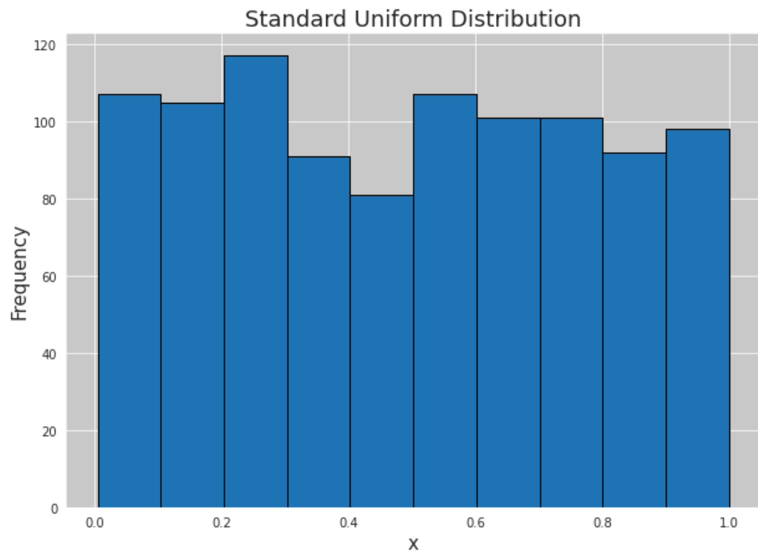

Fig.1

Fig.2 is the Uniform Distribution Plot by generating random numbers for scaled y values.

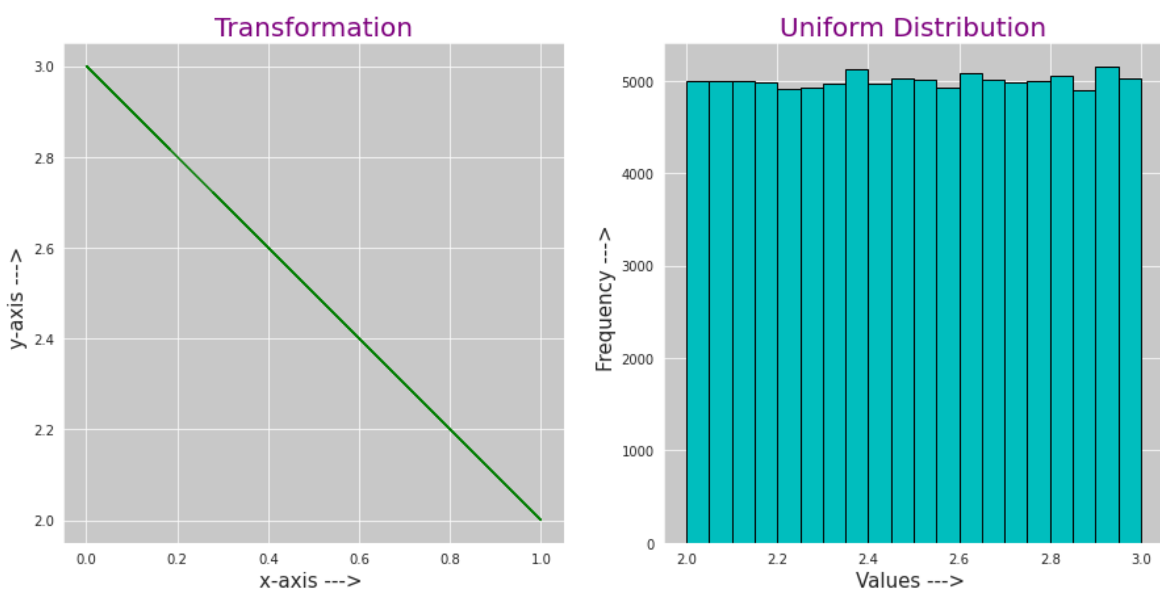

Fig.2

b. Normal Distribution:

Also known as a bell curve or Gaussian distribution is a probability distribution that is symmetric about the mean, showing that data near the mean is more frequent in occurrence than data far from the mean.

The normal distribution model is motivated by the Central Limit Theorem.

The probability density function of a normal distribution is as follows:

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\left(-0.5\left(\frac{(x-\mu)}{\sigma}\right)^2\right)}$$

Here, σ is the standard deviation and μ the mean of the distribution.

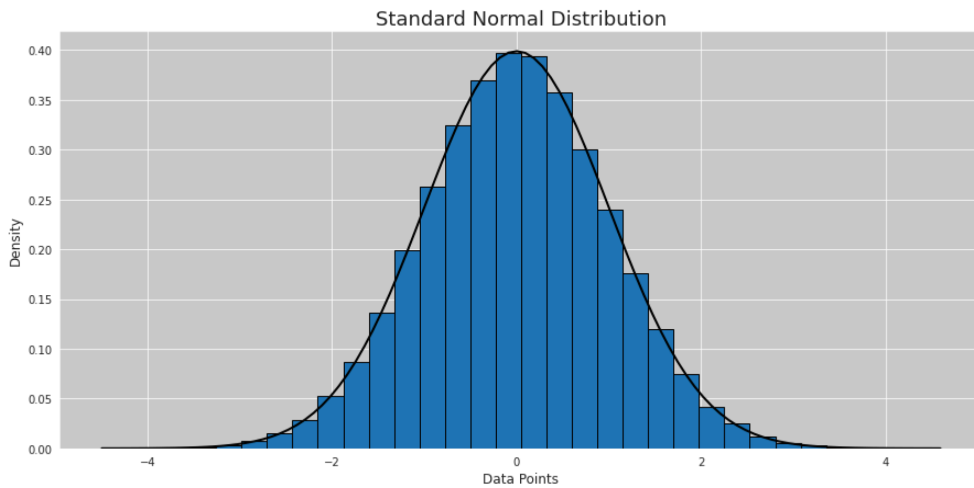

Fig.3

From the graph, we can conclude that the curve is symmetric around the mean and the total area under the curve is 1.
68% of the data falls within one standard deviation of the mean.
95% of the data falls within two standard deviation of the mean.
99.7% of the data falls within three standard deviation of the mean.

2.3 Monte Carlo Method

Monte Carlo Method is a computational algorithm that primarily uses repeated random sampling to obtain results. It uses random numbers to generate deterministic results. The Monte Carlo method is done using approaches (Kroese 2014) like optimization, numerical integration, sampling, posterior distribution (using the Markov Chain Monte Carlo method), and simulations based on probability distributions.

These approaches are based on a rough sequence (Adekitan 2014):

1. We define a domain of possible inputs
2. We randomly generate inputs from domain
3. We perform a deterministic computation using those inputs
4. We aggregate the results to give the final computation

2.4 Applications of Monte Carlo Method:

2.4.1 Estimating the value of pi using Monte Carlo Method:

In this method, we are assigning random variables (from a uniform distribution) to a square in which a circle is inscribed. The value of pi can be found by comparing the total area of the square to the circle within the same bounds.

Now let the circle of side ‘a’ is inscribed inside the square of side ‘2a’.

Area of Square, $A_{square} = (2a)^2 = 4a^2$

Area of circle, $A_{circle} = \pi a^2$

Total random numbers generated = N

Number of points inside the circle (N_c) $\propto A_{circle}$

Number of points inside the square (N_s) $\propto A_{square}$

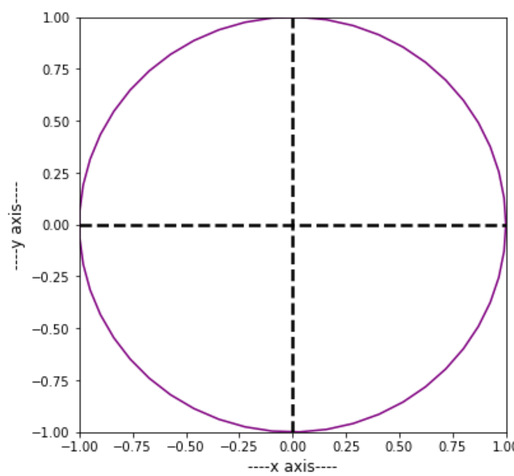

Fig.4

$$\frac{N_C}{N_S} = \frac{A_{Circle}}{A_{Square}}$$

$$\frac{N_C}{N_S} = \frac{\pi a^2}{4a^2}$$

$$\frac{N_C}{N_S} = \frac{\pi}{4}$$

Therefore, $\pi = 4\left(\frac{N_C}{N_S}\right)$

In the program, we are generating random points (from the uniform distribution library) and taking one-fourth of the square and circle and then the value of pi is calculated using the number of points inside the square and the circle.

(Appendix 1.1)

Enter the value of N = 5000
Radius of the circle = 1
Value of pi using Monte Carlo Simulation = 3.1272

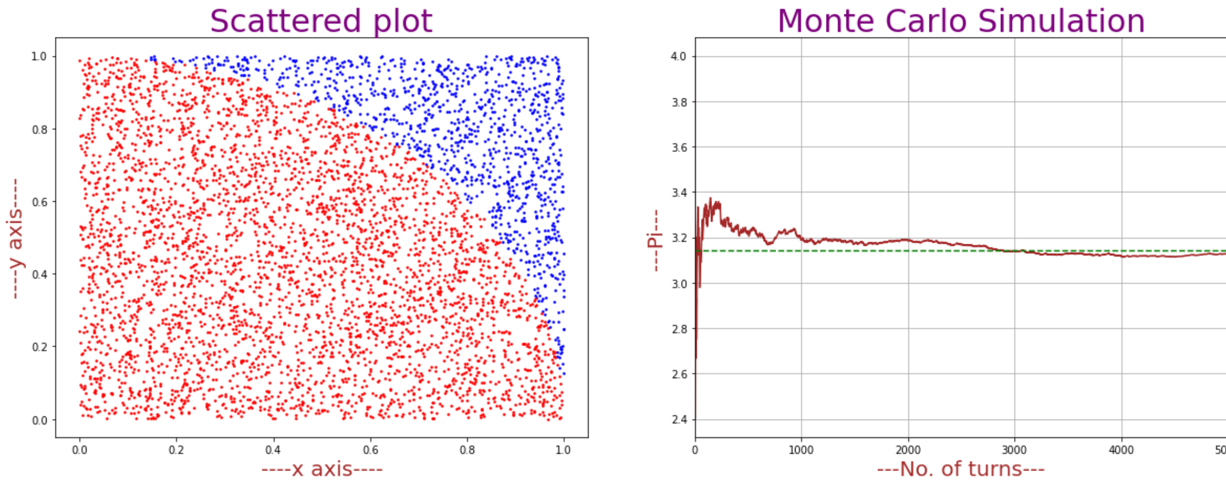

Fig.5

The value of pi here is 3.1272 whereas the actual value of pi is 3.14. We can increase the accuracy of pi calculated by increasing the number of random points generated.

We can use a similar analysis for spheres and cubes to get the value of pi.
(Appendix 1.2)

$$V_{sphere} = \frac{4}{3}\pi r^3 = \frac{4}{3}\pi\left(\frac{a}{2}\right)^3$$

$$V_{cube} = a^3$$

$$\frac{V_{sphere}}{V_{cube}} = \frac{N_S}{N_C} = \frac{\pi}{6}$$

$$\pi = 6 \times \left(\frac{N_S}{N_C}\right)$$

Enter the value of N: 10000
Radius of the circle = 1
Value of pi using Monte Carlo Simulation = 3.1152

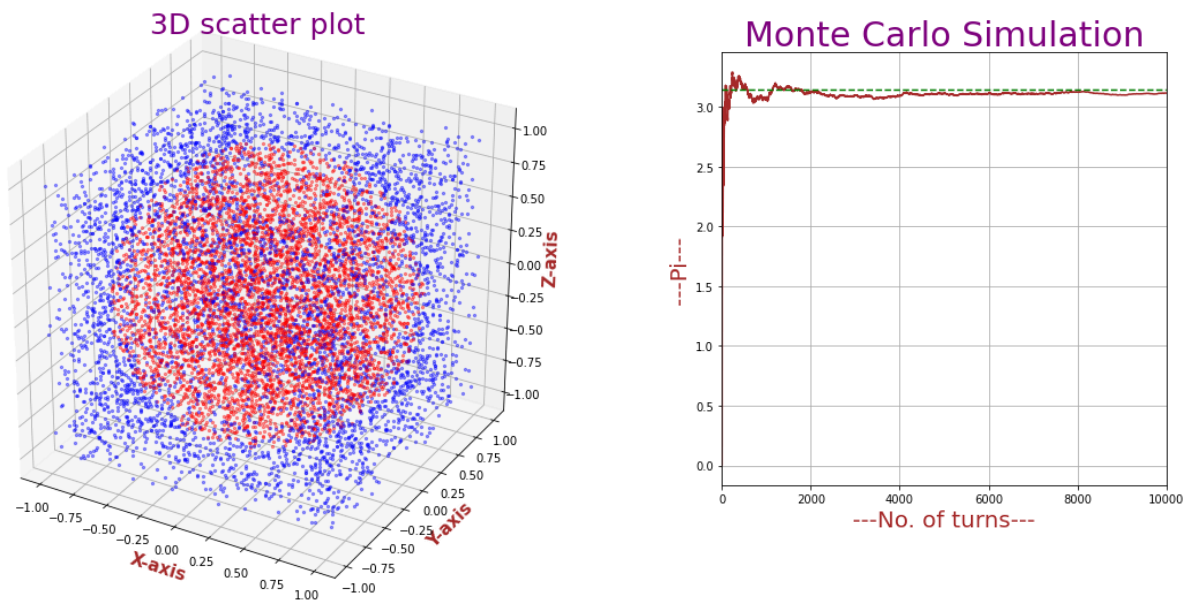

Fig.6

Like in the case of circle-square, the accuracy in estimation of pi increases as the number of points are increased.

2.4.2 Integration or Estimation of the area under the curve:

1. Hit and Miss Method:

In this method of integration, we can find the area under the given curve by generating random numbers for the function f(x) between the interval [a,b] on the abscissa and [f(a),f(b)] on ordinate. Then we find the ratio between the points lying under or on the curve and the total number of points generated.

a. Univariate:

Let the Area of the rectangle be A and A_c be the area under the curve. Let the total number of points generated be N and points lying on and under the curve be c.

<u>(Appendix 1.3)</u>

$$A = (b - a) \times (\, f(b) - f(a)\,)$$

$$A_c = \int_a^b f(x)$$

$$\frac{A_C}{A} = \frac{c}{N}$$

$$A_C = \frac{c}{N} \times A$$

Total no. of darts inside the contour = 3242
Integral with 3 decimal place accuracy: 4.074017353175243

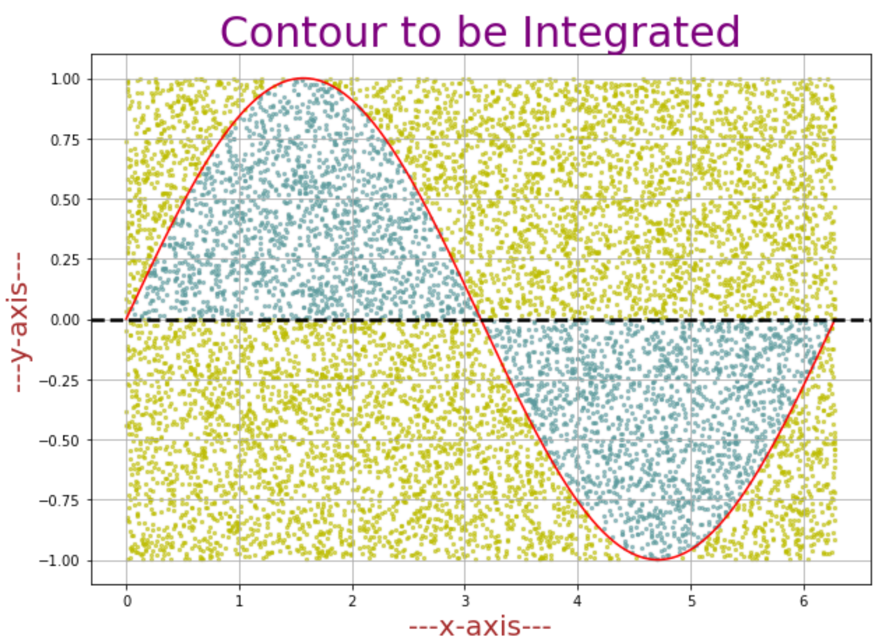

Fig.7

b. Multivariate :

(Appendix 1.4)
Integral= 0.1634

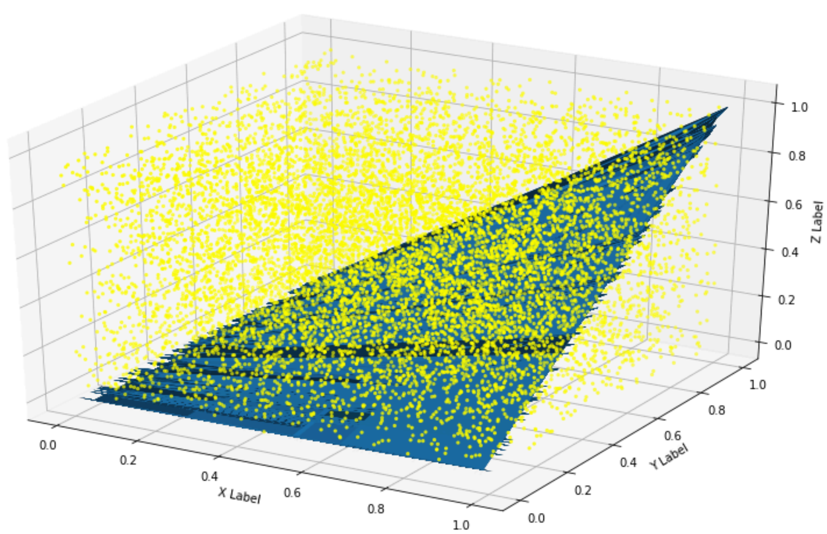

Fig.8

2. Averaging the functions over a large number of intervals

In this method of integration, we divide the area under the curve into a large number of bands, and summing over all the bands gives us the area under the whole curve.

Let the curve lie between point a and b on the x-axis then,

$$f_{avg} = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

$$= \lim_{n\to\infty} \frac{b-a}{n}\left(\frac{1}{b-a}\right)\sum_{i=1}^{n} f(x_i)$$

$$= \frac{1}{b-a}\lim_{n\to\infty} \Delta x \sum_{i=1}^{n} f(x_i)$$

$$= \frac{1}{b-a}\int_a^b f(x)\,dx$$

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} f(x_i) = \frac{1}{b-a}\int_a^b f(x)\,dx$$

$$\int_a^b f(x)\,dx = (b-a)\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

2.4.3 Calculating the cost of pudding using Monte Carlo Method:

In this problem, we first generated the random price for each ingredient that was used in the recipe of Pudding in the given price range. The cost of each ingredient was considered as a parameter.

Ingredients:

1. 2 cups of milk
2. 2 eggs
3. 100 gm sugar
4. 2 slices of bread
5. 100 gm of almonds

Ingredient	Distribution	Parameters Of the distribution
Milk	Uniform	18/lt to 20/lt
Egg	Discrete	1) 20% Rs 2 2) 50% Rs 2.5 3) 30% Rs 3
Sugar	Uniform	20/kg to 23/kg
Bread	Normal	For 12 slices,μ = Rs 25, σ = Rs 1
Almonds	Discrete	1) 70% Rs 500/kg 2) 30% Rs 700/kg

For 1000 different prices of ingredients there are 1000 different prices of pudding. Then, the mean price of pudding is calculated.
(Appendix 1.5)

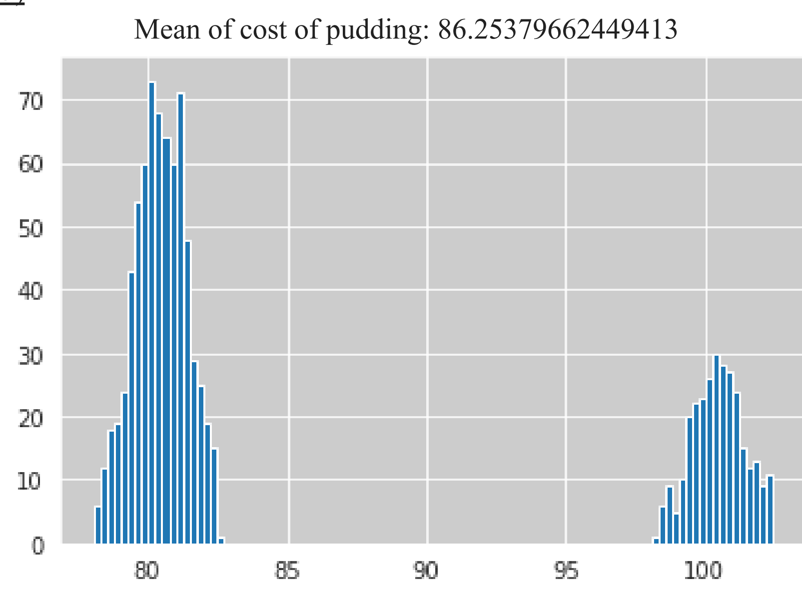

Fig.9

We have also plotted a correlogram (A graph that shows the correlation between two variables) from which we can see that the cost of almonds is affecting the overall price of pudding. Hence, the cost of pudding depends primarily upon the cost of almonds.

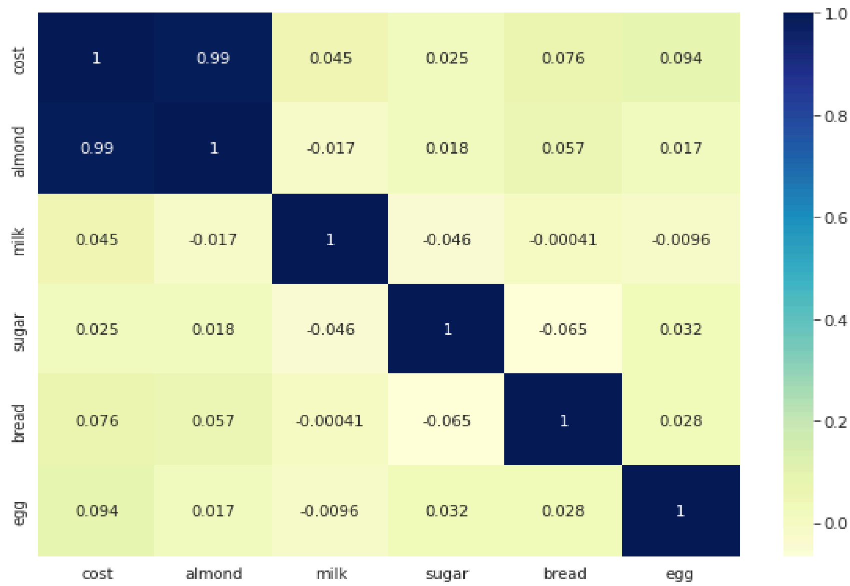

Fig.10

2.5 Markov Chain Monte Carlo

Markov Chain Monte Carlo sampling is an algorithm used for systematic random sampling from high-dimensional probability distributions. Unlike Monte Carlo sampling methods which are able to draw samples independent from each other from the distribution, Markov Chain Monte Carlo methods draw samples where the next sample is dependent on the existing sample which is called a Markov Chain. This property of MCMC allows the algorithms to narrow in on the quantity that is being approximated from the distribution, even with a large number of random variables.

Example:- Generate Sample data for the following probability density function.

$$\left| \frac{cos(x^2)\ (21+sin(x+3))}{exp(x^2)\ exp(-x)} \right|$$

(Appendix 1.6)

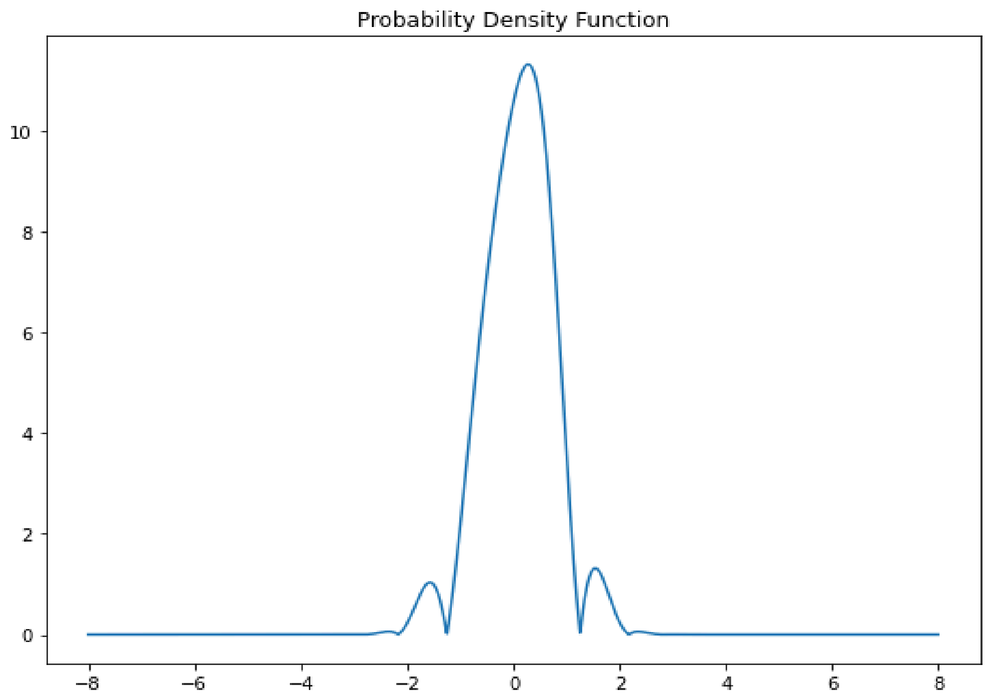

Fig.11

Now, generating sample data using Markov Chain Method with only 1 walker collecting samples proportional to the probability distribution function. Fig. 12 is the data generated using 1 walker only.

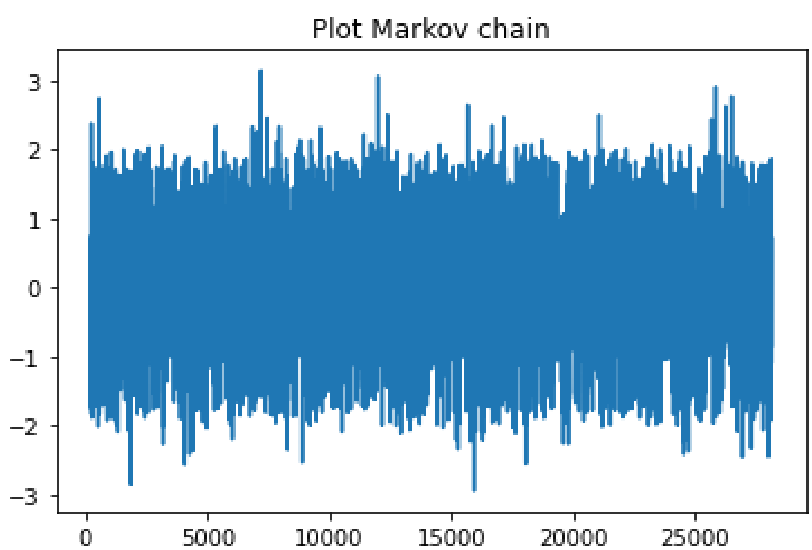

Fig.12

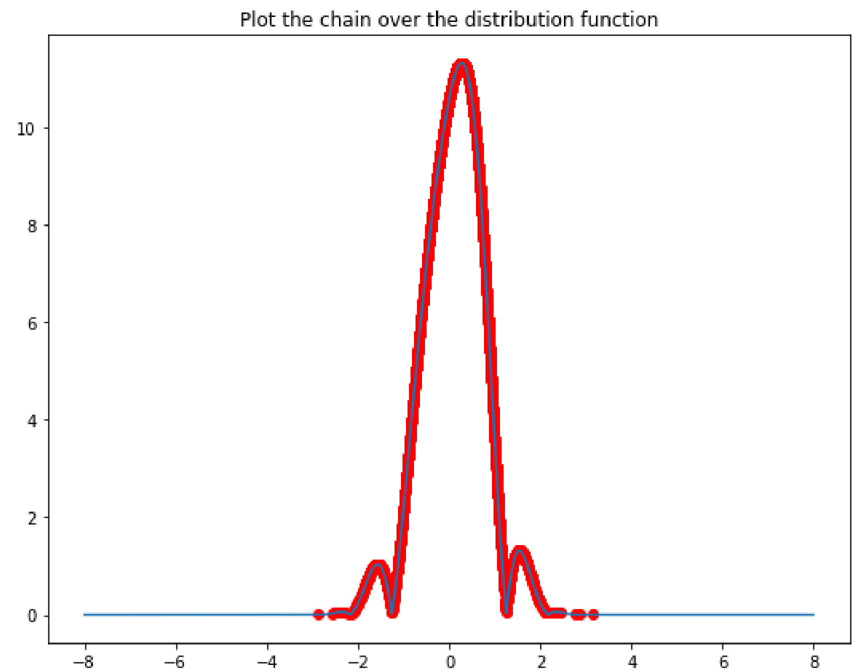

Fig.13

Fig shown above is the illustration of how well our estimated chain using 1 walker fits over the original distribution function.

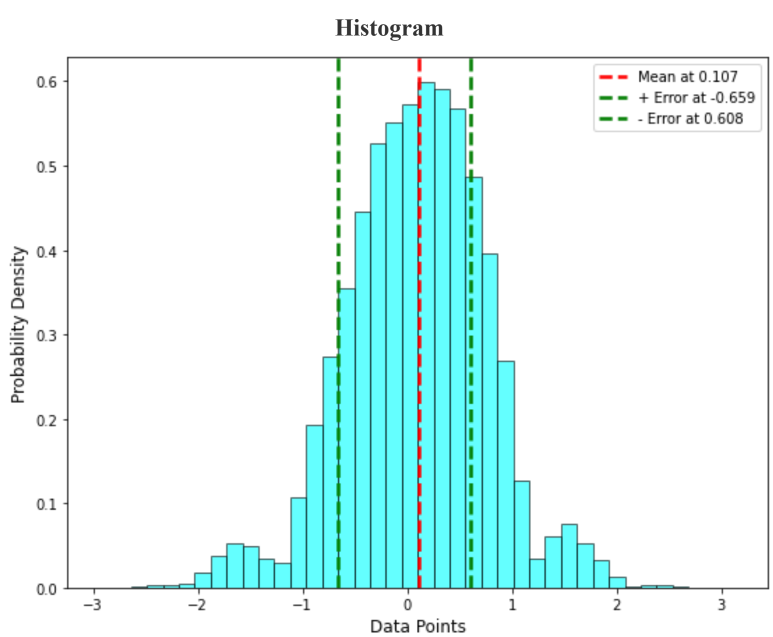

Fig.14

Now, Generating sample data using Markov chain Method with multiple walkers each having a different chain.

Plot Markov Chain

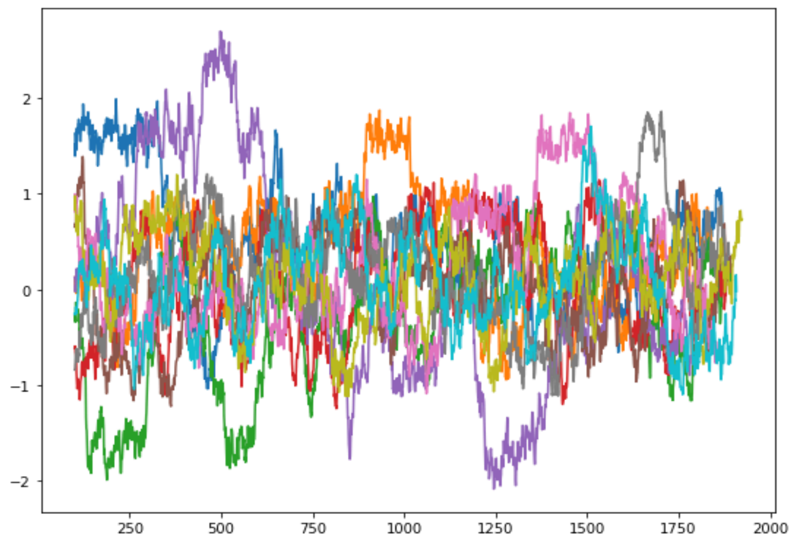

Fig.15

Histogram

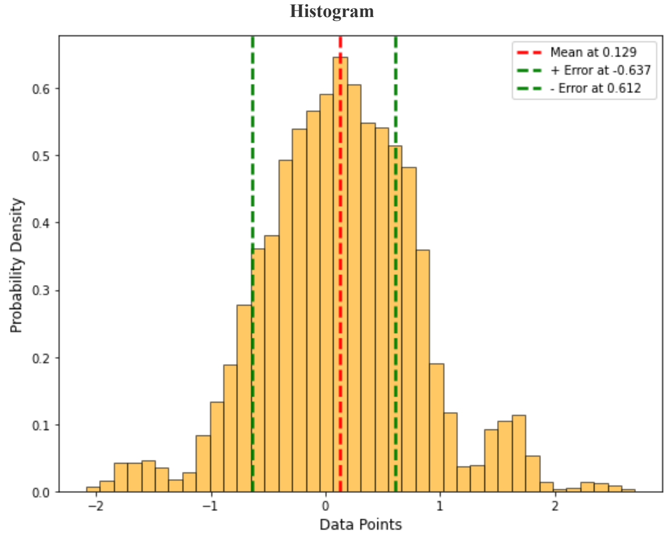

Fig.16

2.6 Maximum Likelihood

For a chosen statistical model, the likelihood function describes (Berger and Casella 2002) the joint probability of the observed data as a function of the parameters.

Consider a parameter value $\boldsymbol{\theta}$ in a parametric space $\boldsymbol{\Theta}$. We define $\mathbf{p}(\mathbf{X}|\boldsymbol{\theta})$ as the likelihood function that gives the probabilistic prediction for the data set **X**.

The likelihood principle (Wolpert and Berger 1988) states that, given a statistical model, all the information X provides about $\boldsymbol{\theta}$ gives the likelihood function.

2.7 Chi-Square Method

Chi Square Fitting is used to determine whether a variable is likely to come from a specified distribution or not.

Formula: $$\chi^2 = \sum_{i=1}^{n} \frac{(observed - expected)^2}{expected}$$

2.6.1 Chi-Square Fitting using one parameter

We are given the data with z, H(z) and σ_H where each data point is independent and normally distributed.

z (Redshift)	H(z) (Hubble Parameter)	σ_H (Error Bar)
0.0907	69	12
0.17	83	8
0.179	75	4
0.199	75	5
0.24	79.69	2.65
0.27	77	14
0.352	83	14
0.4	95	17
0.43	86.45	3.68

0.48	97	62
0.593	104	13
0.68	92	8
0.781	105	12
0.875	125	17
0.88	90	14
0.9	117	23
1.037	154	20
1.3	168	17
1.43	177	18
1.53	140	14
1.75	202	40

Theoretical model:

$H^{th}(z) = H_o\sqrt{(\Omega_m(1 + z)^3 + \Omega_\Lambda)}$

Where H_o=67.8

Using this model we calculated the most likely value of model parameters i.e. Ω_m and Ω_Λ for this data to occur by constraining the value of Ω_m in the range 0 to 1 and taking Ω_Λ = 1-Ω_m .

By Chi

$$\chi^2 = \sum_{i=1}^{n} \frac{\left(H_{obs,i} - H_{th,i}\right)^2}{\sigma_i^2}$$

where, H_{obs}, H_{th} and σ are the values observed, theoretical values and the standard deviation respectively.

(Appendix 1.7)

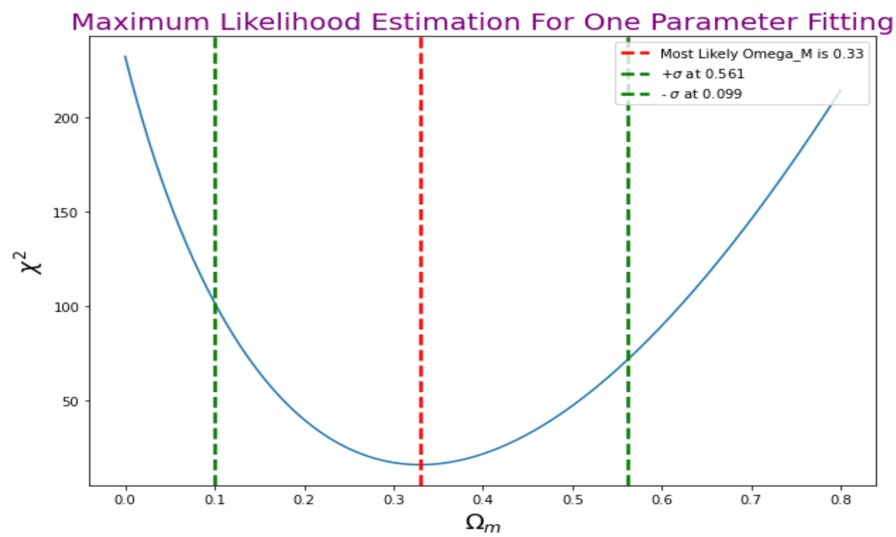

Fig.17

2.6.2 Chi-Square Fitting using one parameter

Using the same model, now both model parameters i.e. Ω_m and Ω_Λ are unknown so using Two Parameter Fitting the most likely value of model parameters is being calculated.

(Appendix 1.8)

Chi-Square is minimum at: 15.364290464424073
Omega matter: 0.3083999
Omega Lambda: 0.7554

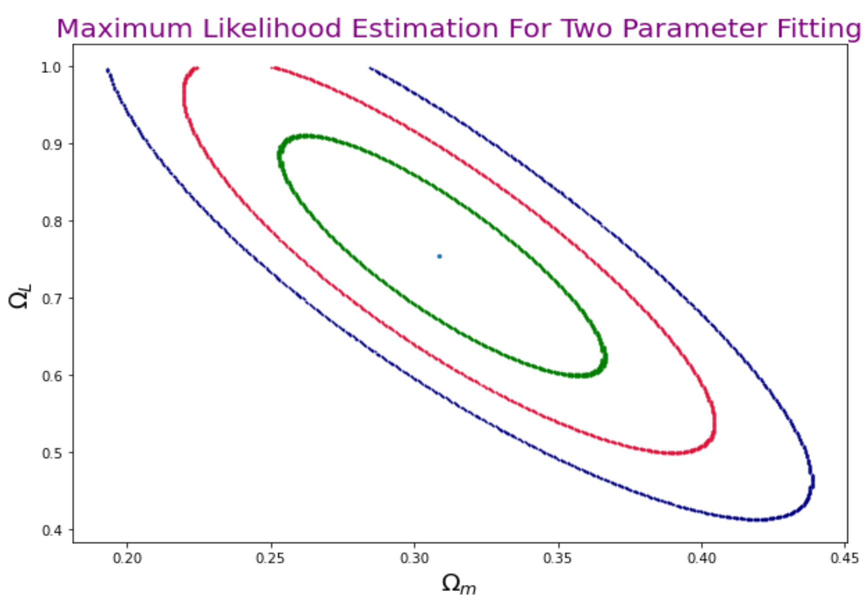

Fig.18

CHAPTER-III: RESEARCH PAPER

Fast Radio Bursts (FRBs) are one of the most mysterious entities in astrophysics which are intense, bright (50 mJy-100 mJy) pulses of radio frequency emissions, lasting on the order of milliseconds.They have high Galactic latitudes and excessive dispersion measures (DMs), which point to a cosmic origin for these enigmatic transients.

Dispersion Measure (DM) is the measure of the amount of dispersion suffered by radio waves from an astronomical object. It depends on the electron density along the line of sight and is conventionally measured in parsecs per cubic centimeter. Dispersion measure is effectively obtained from accurate timing of pulsars and is a key approach of estimating the electron density in interstellar space.

If redshifts of FRBs can be measured, one may combine the DM and z information to study cosmology.

The observed DM of FRB is given by

$$DM_{obs} = DM_{MW} + DM_{IGM} + DM_{HG} \qquad (a)$$

where DM_{MW} ,DM_{IGM}, DM_{HG} denote contribution from the Milky Way, intergalactic medium and FRB host galaxy.

DM_{MW} is strongly correlated with Galactic latitude |b| and can be well constrained by Galactic pulsar data (Taylor & Cordes 1993).With reasonable certainty ,DM_{MW} can be retrieved for well localized FRBs. The extragalactic DM of a FRB is then defined as

$$DM_E \equiv DM_{obs} - DM_{IGM} = DM_{IGM} + DM_{HG} \qquad (b)$$

ΛCDM (Lambda Cold Dark Matter) Model is a cosmological model that describes our universe in terms of expansion, matter, and energy content and the relationship between them. It tells us how long ago and at what distance the cosmological background radiation was created.

Since **ΛCDM** is consistent with essentially all the observational constraints, we focus on this model with $\Omega_\Lambda + \Omega_m = 1$ enforced.

We define the mean DM of the IGM, which is given by taking into account the local inhomogeneity of the IGM (Deng & Zhang 2014)

Given, $$K_{IGM} = \frac{3cH_0\Omega_b f_{IGM}}{8\pi G m_p} \qquad (1.)$$

$$<DM_{IGM}> = K_{IGM}\int_0^z \frac{f_e(z')\,(1+z')\,dz'}{\sqrt{\Omega_m(1+z')^3+\Omega_\Lambda}} \tag{2.}$$

Where H_o is current Hubble constant, Ω_b is current baryon mass density fraction of the universe, f_{IGM} is the fraction of baryon mass in the IGM, $f_e(z)$ =(¾) $y_1\chi_{e,H}(z)$ + (1/8) $y_2\chi_{e,He}(z)$,y_1 ~1 and $y_2 \simeq 4 - 3y_1$ are the hydrogen and helium mass fractions normalized to 3/4 and 1/4, respectively, and $\chi_{e,H}(z)$ and $\chi_{e,He}(z)$ are the ionization fractions for hydrogen and helium, respectively. For FRBs at z < 3, both hydrogen and helium are fully ionized (Meiksin 2009; Becker et al. 2011). One then has $\chi_{e,He}(z)=\chi_{e,H}(z) = 1$,
$f_e(z) \simeq 7/8$.

Since $f_e(z) \simeq 7/8$ for z < 3, α essentially depends only on the cosmological parameters (Ω_m, Ω_Λ). The $<DM_{IGM}>$–z relation and α as a function of z are presented in Fig. 19 for Ω_m = 0.1, 0.3, 0.5, respectively. One can see that α is around 1 at z<1. It initially rises and monotonically decreases with z after reaching a peak.

DM_{IGM} cannot be directly measured observationally; hence, cannot be directly measured. DM_E and z are the parameters that can be directly monitored, thus we introduce a slope parameter

$$\beta(z) \equiv \frac{d\ln\langle \mathrm{DM_E}\rangle}{d\ln z}$$
$$= \frac{z}{\langle \mathrm{DM_E}\rangle}\left(\frac{d\,\langle \mathrm{DM_{IGM}}\rangle}{dz} + \frac{d\,\langle \mathrm{DM_{HG}}\rangle}{dz}\right)$$

For a host galaxy at redshift z, due to cosmological redshift and time dilation, its observed DM_{HG} is a factor of 1/(1 + z) of the local one $DM_{HG,loc}$ (Ioka 2003; Deng & Zhang 2014).

(Appendix 1.9)

(a) $<DM_{IGM}> - z$ relation (b) $\alpha - z$ relation

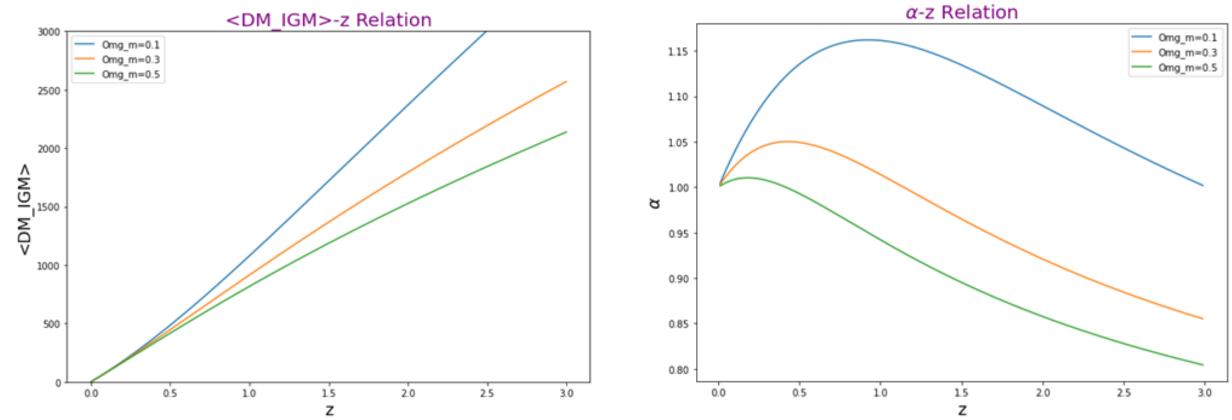

Fig.19

(a) $<DM_{IGM}>$–z relation. We adopted the best-constrained values of the following parameters (Planck Collaboration et al. 2016): H0 = 67.7 km s Mpc , Ω_m = 0.049, f_{IGM}=0.83 (b) α–z relation. The blue, red, and yellow lines denote Ω_m = 0.1, 0.3, 0.5, respectively.

(Appendix 2.0)

(a) $<DM_E> - z$ relation (b) $\beta - z$ relation

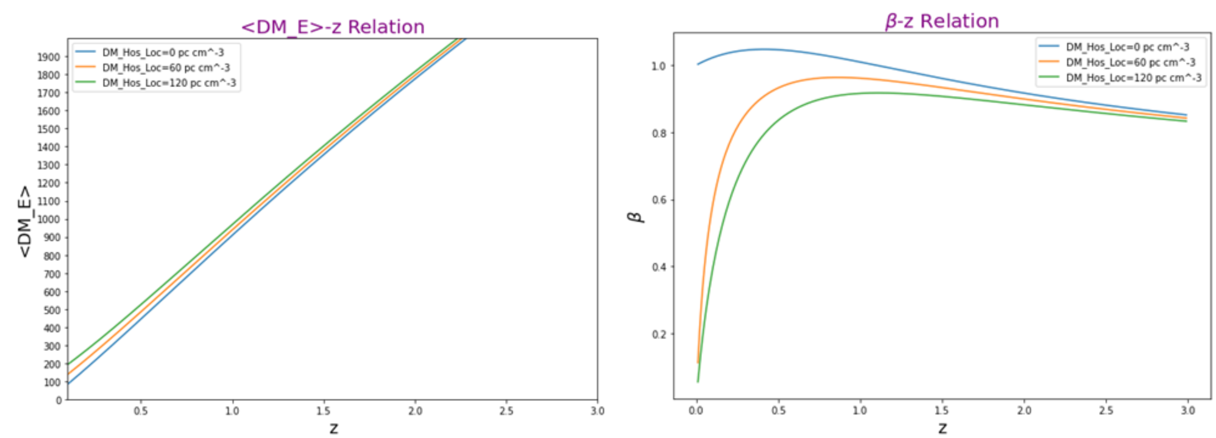

Fig.20

Fig.20 (a) $<DM_{IGM}>$–z relation. We adopt the same parameters as Fig. 19 and Ω_m = 0.31. (b) β–z relation. The blue, red, and yellow lines denote $<DM_{HG,loc}>$ = 0, 60, 120 pc cm^{-3}, respectively.

Now we demonstrate via Monte Carlo simulations that the average value of the local host galaxy DM, $DM_{<HG,loc>}$ along with other cosmological parameters (such as the mass density Ω_m in the ΛCDM model and the IGM component of the baryon energy density $\Omega_b f_{IGM}$) can be measured independently using minimum chi square fitting to the data.

3.1 Generation of Mock Data Sample

The three unknown parameters, $DM_{HG,loc}$, Ω_m and K_{IGM} are defined by properties of log DM_E – log z plot, and therefore can be independently inferred from ($<DM_E>$, z) data of samples of FRBs.

To prove this, we applied Monte Carlo Simulations to show that minimum chi square fitting can be used to infer the three unknown parameters.

$$DM_E = DM_{IGM} + \frac{DM_{HG,loc}}{1+z} \quad (3.)$$

$$\chi^2(\Omega_m, < DM_{HG,loc} >, K_{IGM}) = \Sigma \frac{(DM_{E,i} - <DM_E>)^2}{\sigma^2_{IGM,i} + [\sigma_{HG,loc,i}/(1+z_i)]^2} \quad (4.)$$

For generating the samples, we adopted the flat ΛCDM parameters that were recently derived from Planck data: H_o= 67.7 km s^{-1} Mpc^{-1} , Ω_m= 0.31 , Ω_Λ= 0.69, Ω_b= 0.049 (Planck Collaboration et al. 2016). For the fraction of baryon mass in IGM, we adopted f_{IGM} =0.83 (Fukugita et al. 1998; Shull et al. 2012; Deng & Zhang 2014). Putting the values of the above parameters in equation (1.), the value of K_{IGM} was calculated which came out to be 933 pc cm^{-3}.

The generation of synthetic sample has the following procedure:

1. Assuming $P(z)=ze^{-z}$, a phenomenological model for GRB redshift distribution, satisfying the FRBs redshift distribution, an inverse transformation method was used to sample the values of z. Using the probability distribution function, P(z), the cumulative distribution function and then its inverse was calculated. Random numbers were generated between (0,1) using a uniform distribution generator and used as input in the inverse function to generate samples of z.
2. Putting the values of z, K_{IGM}, Ω_m and Ω_Λ in equation (2.), the corresponding values of DM_{IGM} were calculated. To introduce some randomness in the sample of DM_{IGM}, we took Normal Distribution, N(DM_{IGM}, 100 pc cm^{-3}).

3. Normal distribution, $N(100 \text{ pc cm}^{-3}, 20 \text{ pc cm}^{-3})$ was taken to sample the values of $DM_{HG,loc}$.
4. Using the values of $DM_{HG,loc}$, DM_{IGM} and z, corresponding values of DM_E were calculated using the equation (3.).

Using the above procedure, six datasets were generated

1. 50 Samples for redshift between $0<z<3$ and $DM_{HG,loc}=N(100 \text{ pc cm}^{-3}, 20 \text{ pc cm}^{-3})$
2. 500 Samples for redshift between $0<z<3$ and $DM_{HG,loc}=N(100 \text{ pc cm}^{-3}, 20 \text{ pc cm}^{-3})$
3. 500 Samples for redshift between $0<z<2$ and $DM_{HG,loc}=N(100 \text{ pc cm}^{-3}, 20 \text{ pc cm}^{-3})$
4. 500 Samples for redshift between $0<z<1$ and $DM_{HG,loc}=N(100 \text{ pc cm}^{-3}, 20 \text{ pc cm}^{-3})$
5. 500 Samples for redshift between $0<z<3$ and $DM_{HG,loc}=N(100 \text{ pc cm}^{-3}, 50 \text{ pc cm}^{-3})$
6. 500 Samples for redshift between $0<z<3$ and $DM_{HG,loc}=N(200 \text{ pc cm}^{-3}, 50 \text{ pc cm}^{-3})$

1. Graph for 50 Samples of redshift between $0<z<3$ and $DM_{HG,loc}=N(100 \text{ pc cm}^{-3}, 20 \text{ pc cm}^{-3})$

$z_f = 3$

(Appendix 2.1)

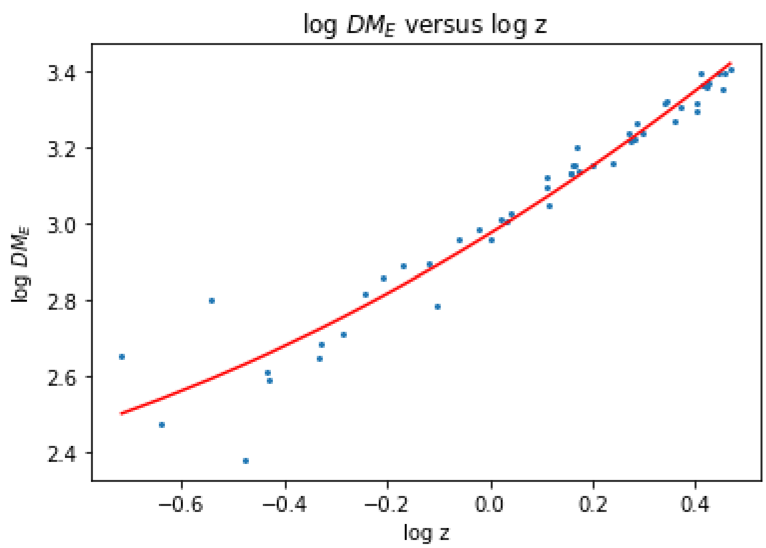

Fig.21

2. Graph for 500 Samples of redshift between $0<z<3$ and $DM_{HG,loc}=N(100 \text{ pc cm}^{-3}, 20 \text{ pc cm}^{-3})$

$z_f = 3$

(Appendix 2.2)

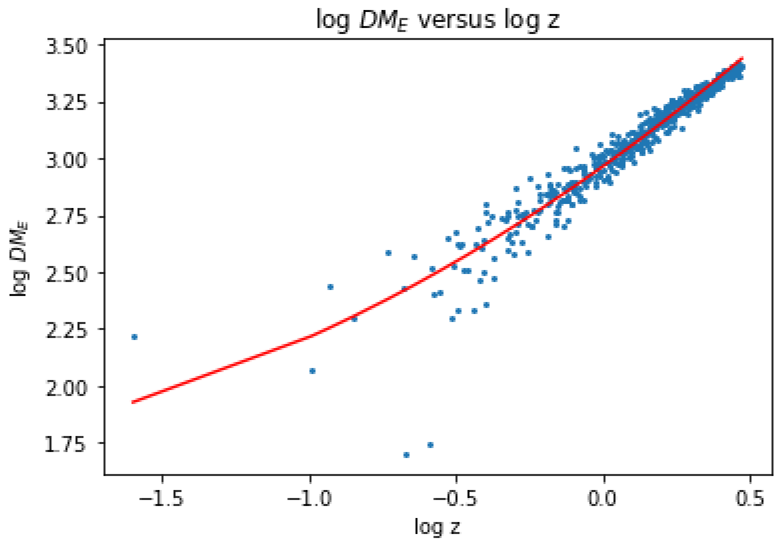

Fig.22

3. Graph for 500 Samples of redshift between 0<z<2 and $DM_{HG,loc}$=N(100 pc cm^{-3}, 20 pc cm^{-3})
   z_f = 2
   <u>(Appendix 2.3)</u>

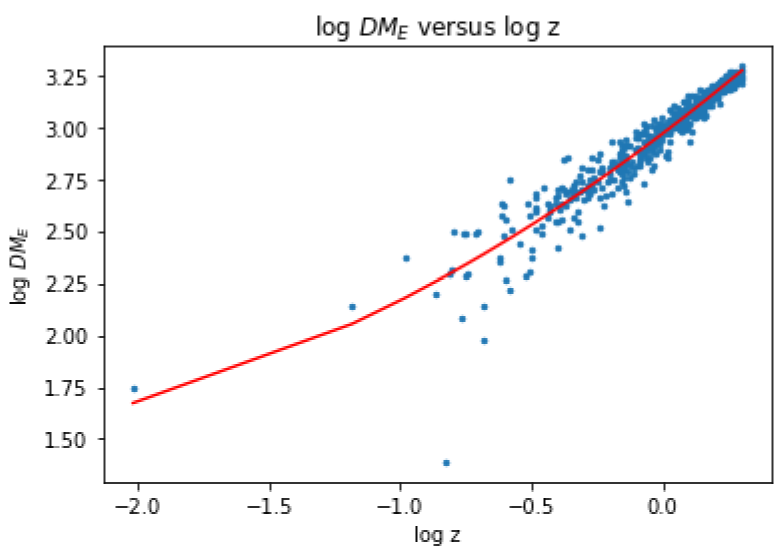

Fig.23

4. Graph for 500 Samples of redshift between 0<z<1 and $DM_{HG,loc}$=N(100 pc cm^{-3}, 20 pc cm^{-3})
   z_f = 1
   <u>(Appendix 2.4)</u>

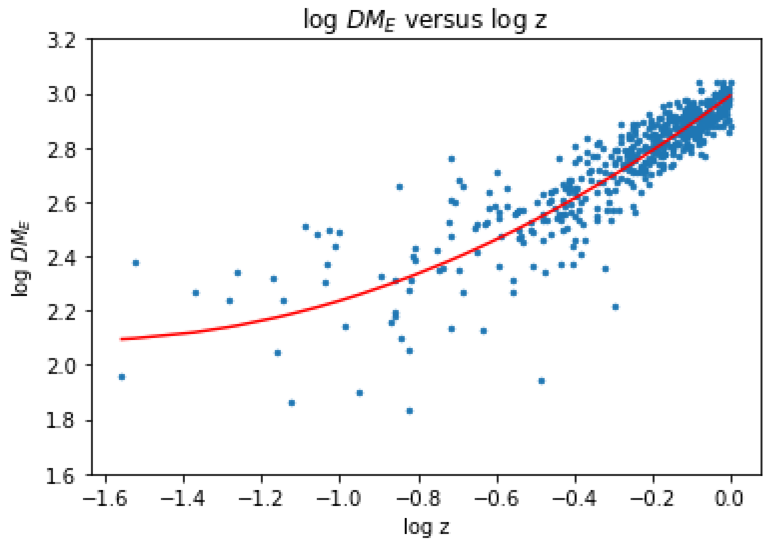

Fig.24

5. Graph for 500 Samples of redshift between 0<z<3 and $DM_{HG,loc}$=N(100 pc cm^{-3}, 50 pc cm^{-3})
(Appendix 2.5)

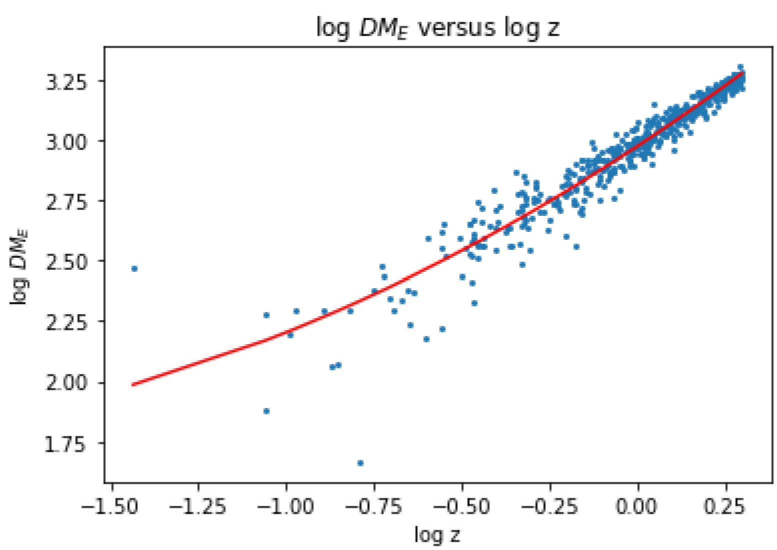

Fig.25

6. Graph for 500 Samples of redshift between 0<z<3 and $DM_{HG,loc}$=N(200 pc cm^{-3}, 50 pc cm^{-3})
(Appendix 2.6)

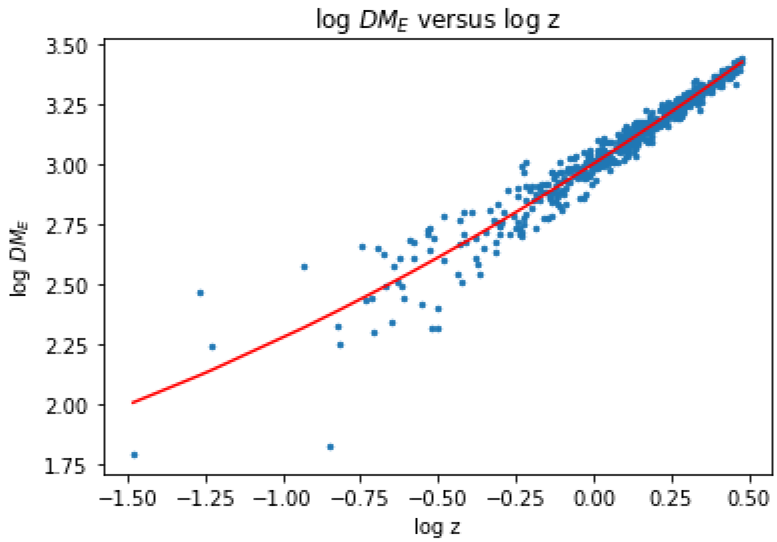

Fig.26

Our next step is to analyze the data and find the chi-square minimum by constraining two parameters at a time and values of the two parameters corresponding to the chi-square minimum.

1. Constraining K_{IGM} and Ω_m and using them to plot 68%, 95% and 98% confidence levels.
2. Constraining K_{IGM} and $DM_{HG,loc}$ and using them to plot 68%, 95% and 98% confidence levels.
3. Constraining $DM_{HG,loc}$ and Ω_m and using them to plot 68%, 95% and 98% confidence levels.

For 50 Samples and $z_f = 3$, we take K_{IGM} in the range 800 to 1100 , constraint Ω_m in the range 0 to 1 and $DM_{HG,loc}$ in the range 50 pc cm^{-3} to 140 pc cm^{-3} .Then finding the best fit using χ^2 minimum taking two parameter constraints at a time . This order of steps is repeated in the sample, for each lens system which is expanded until the minimum chi-square is obtained. We adjust the fits of the model by minimizing the χ^2 function in Equation (4.)

(Appendix 2.7)

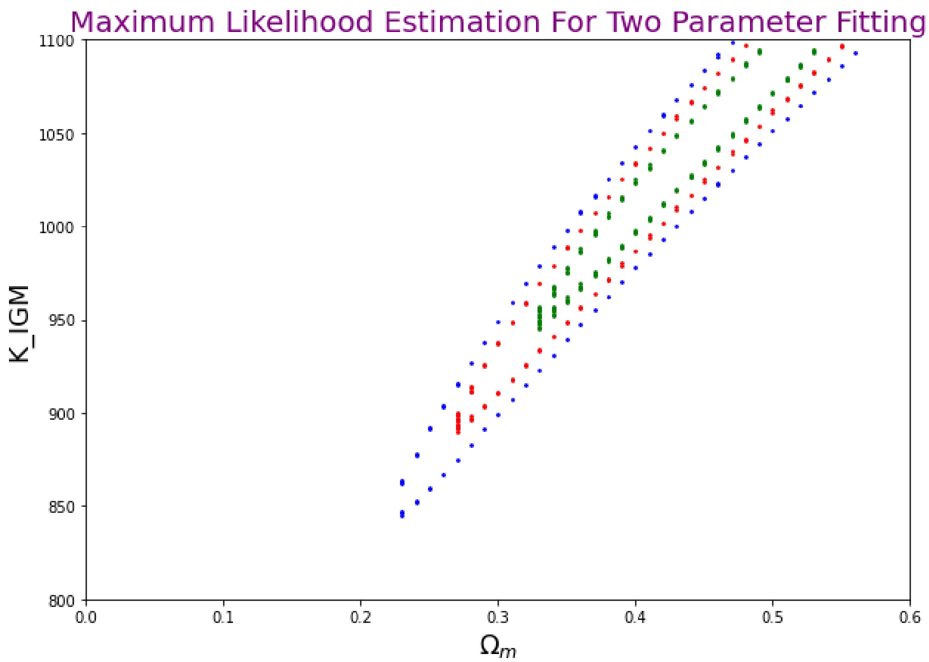

Fig.27

(Constraining K_{IGM} and Ω_m, we obtain a value of 56.888 as the chi-square minimum and the chi-square per degree of freedom is 1.137 for a K_{IGM} value of 1049 and Ω_m value of 0.44)

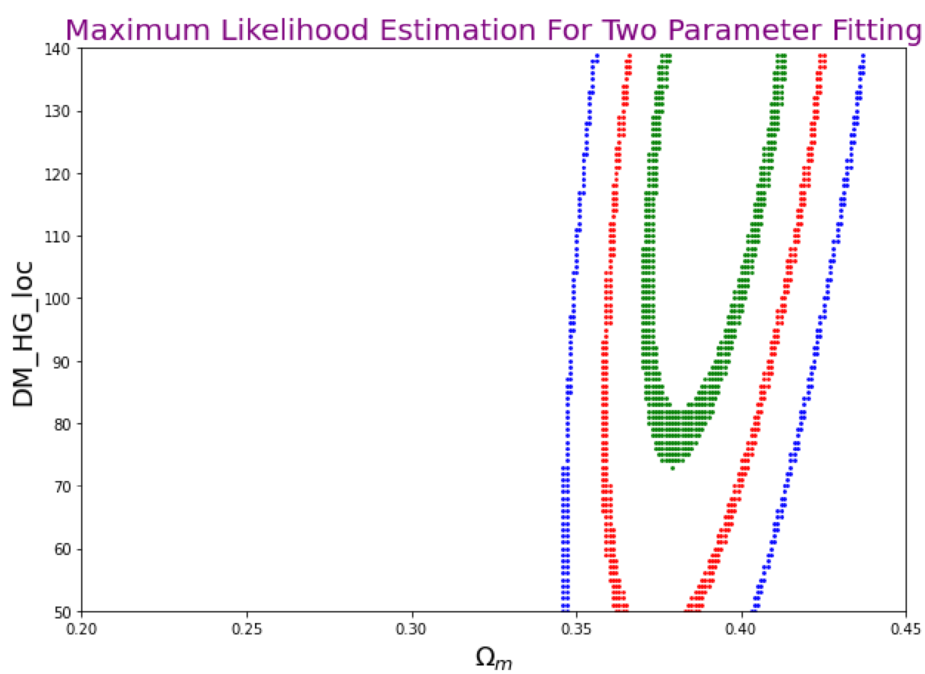

Fig.28

(Constraining $DM_{HG,loc}$ and Ω_m, we obtain a value of 55.239 as the chi-square minimum and the chi-square per degree of freedom is 1.10 for a $DM_{HG,loc}$ value of 131 pc cm^{-3} and Ω_m value of 0.392)

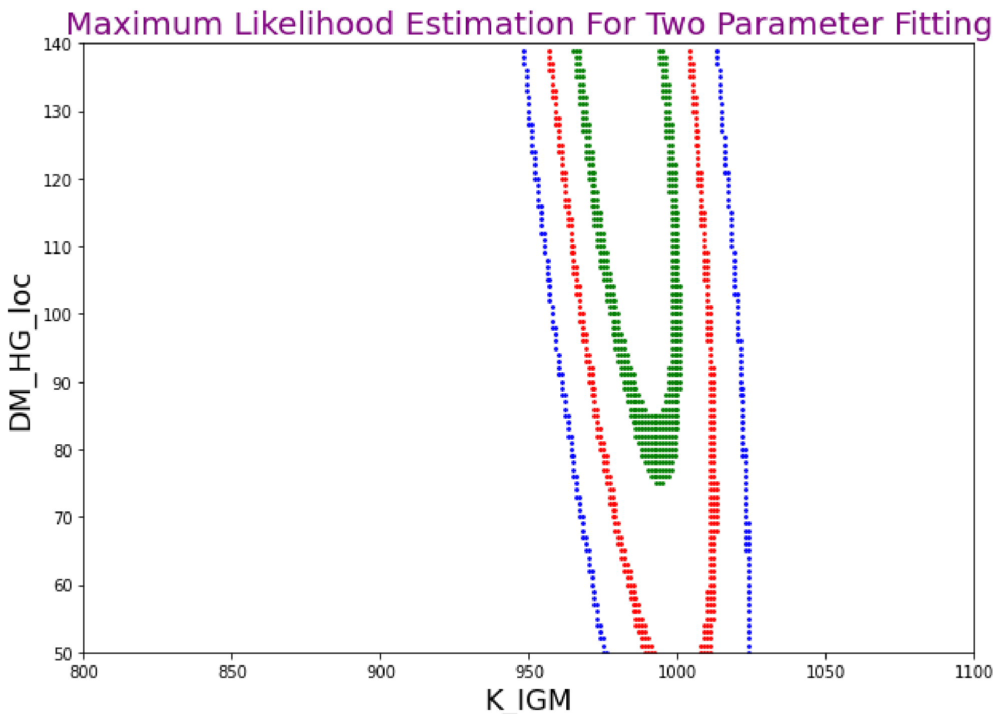

Fig.29

(Constraining $DM_{HG,loc}$ and K_{IGM}, we obtain a value of 55.057 as the chi-square minimum and the chi-square per degree of freedom is 1.101 for a $DM_{HG,loc}$ value of 139 pc cm^{-3} and K_{IGM} value of 980)

For 500 Samples and $DM_{HG,loc}$=N(100 pc cm^{-3}, 20 pc cm^{-3}), we take K_{IGM} in the range 800 to 1100 , constraint Ω_m in the range 0 to 1 and $DM_{HG,loc}$ in the range 60 pc cm^{-3} to 120 pc cm^{-3} .

Then finding the best fit using χ^2 minimum taking two parameter constraints at a time . This order of steps is repeated in the sample, for each lens system which is expanded until the minimum chi-square is obtained.

We adjust the fits of the model by minimizing the χ^2 function in Equation (4.)

For $z_f = 1$ (Appendix 2.8)

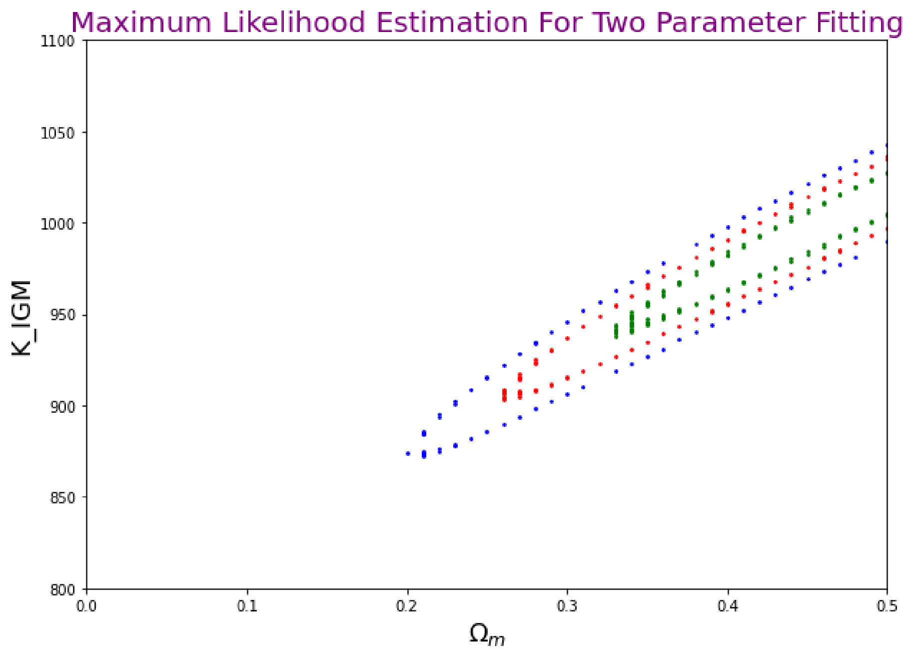

Fig.30

(Constraining K_{IGM} and Ω_m, we obtain a value of 494.436 as the chi-square minimum and the chi-square per degree of freedom is 0.988 for a K_{IGM} value of 1008 and Ω_m value of 0.48)

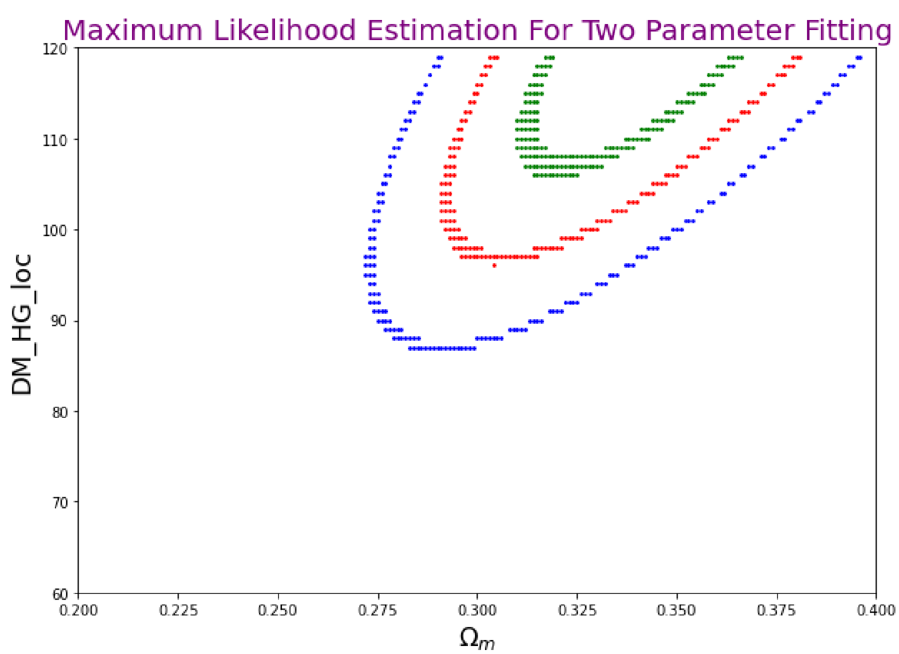

Fig.31

(Constraining $DM_{HG,loc}$ and Ω_m, we obtain a value of 494.082 as the chi-square minimum and the chi-square per degree of freedom is 0.988 for a $DM_{HG,loc}$ value of 119 pc cm^{-3} and Ω_m value of 0.341)

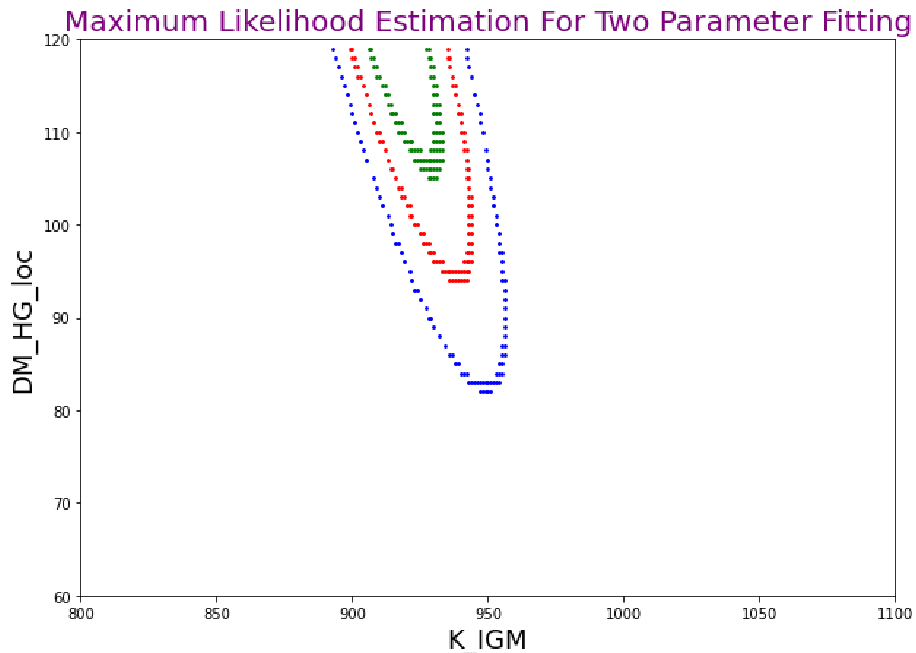

Fig.32

(Constraining $DM_{HG,loc}$ and K_{IGM}, we obtain a value of 494.520 as the chi-square minimum and the chi-square per degree of freedom is 0.980 for a $DM_{HG,loc}$ value of 119 pc cm^{-3} and K_{IGM} value of 917)

For $z_f = 2$ (Appendix 2.9)

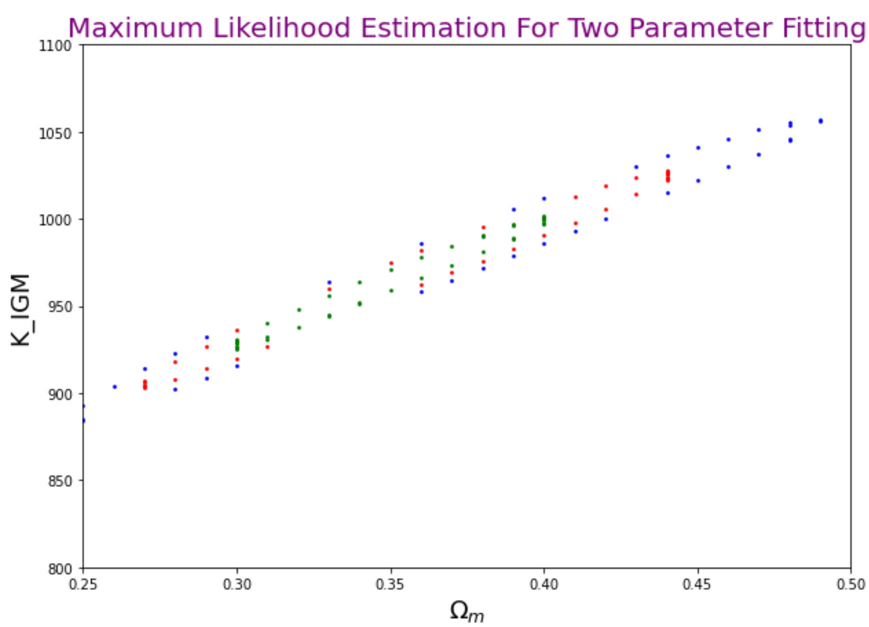

Fig.33

(Constraining K_{IGM} and Ω_m, we obtain a value of 527.885 as the chi-square minimum and the chi-square per degree of freedom is 1.05 for a K_{IGM} value of 965 and Ω_m value of 0.35)

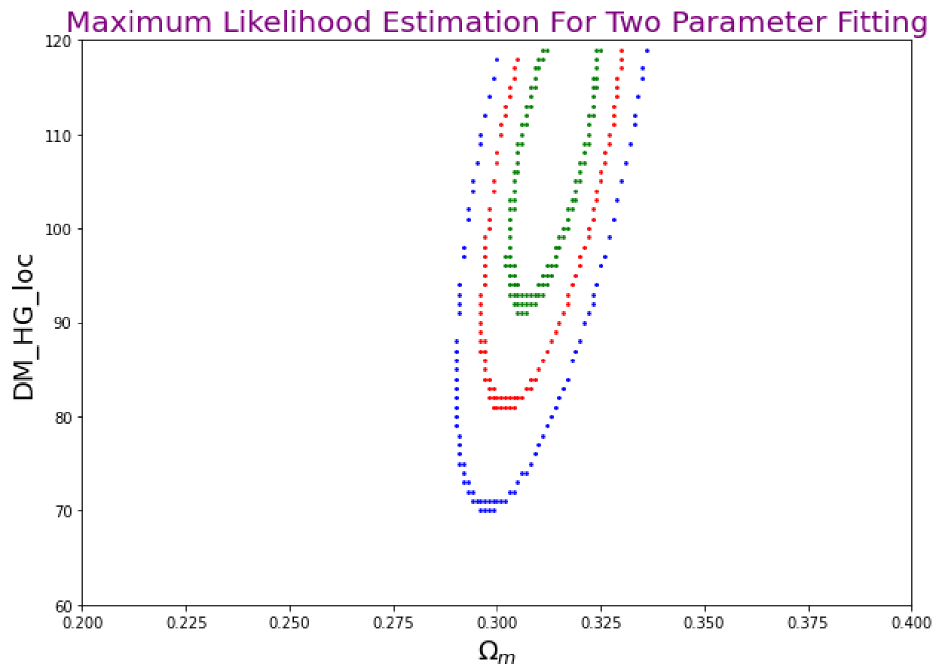

Fig.34

(Constraining $DM_{HG,loc}$ and Ω_m, we obtain a value of 527.365 as the chi-square minimum and the chi-square per degree of freedom is 1.054 for a $DM_{HG,loc}$ value of 109 pc cm^{-3} and Ω_m value of 0.313)

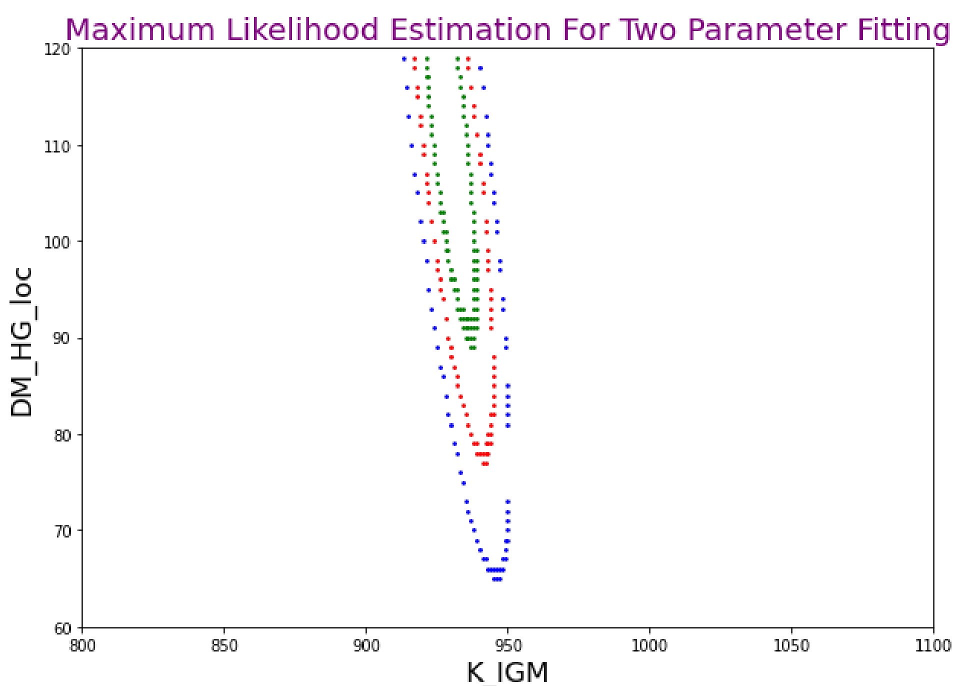

Fig.35

(Constraining $DM_{HG,loc}$ and K_{IGM}, we obtain a value of 527.370 as the chi-square minimum and the chi-square per degree of freedom is 1.054 for a $DM_{HG,loc}$ value of 109 pc cm^{-3} and K_{IGM} value of 930)

For $z_f = 3$ (Appendix 3.0)

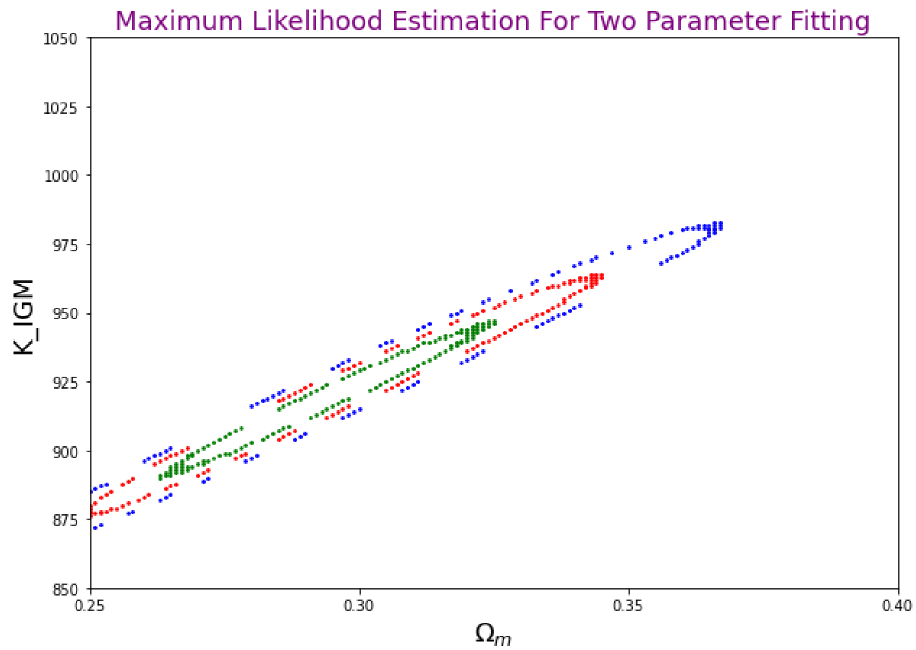

Fig.36

(Constraining K_{IGM} and Ω_m, we obtain a value of 574.056 as the chi-square minimum and the chi-square per degree of freedom is 1.148 for a K_{IGM} value of 917 and Ω_m value of 0.291)

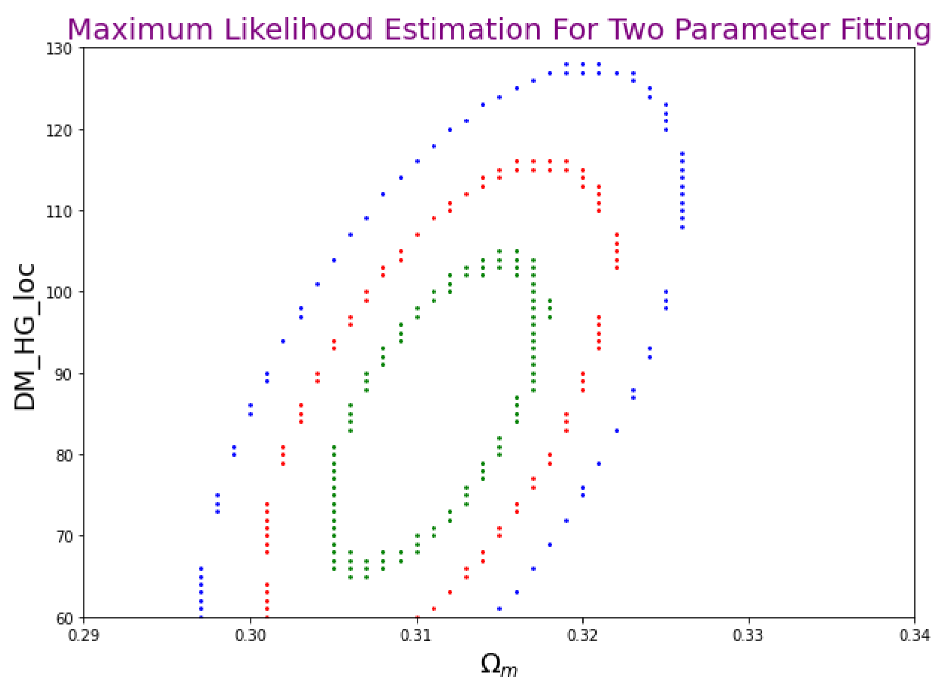

Fig.37

(Constraining $DM_{HG,loc}$ and Ω_m, we obtain a value of 574.466 as the chi-square minimum and the chi-square per degree of freedom is 1.148 for a $DM_{HG,loc}$ value of 85 pc cm^{-3} and Ω_m value of 0.311)

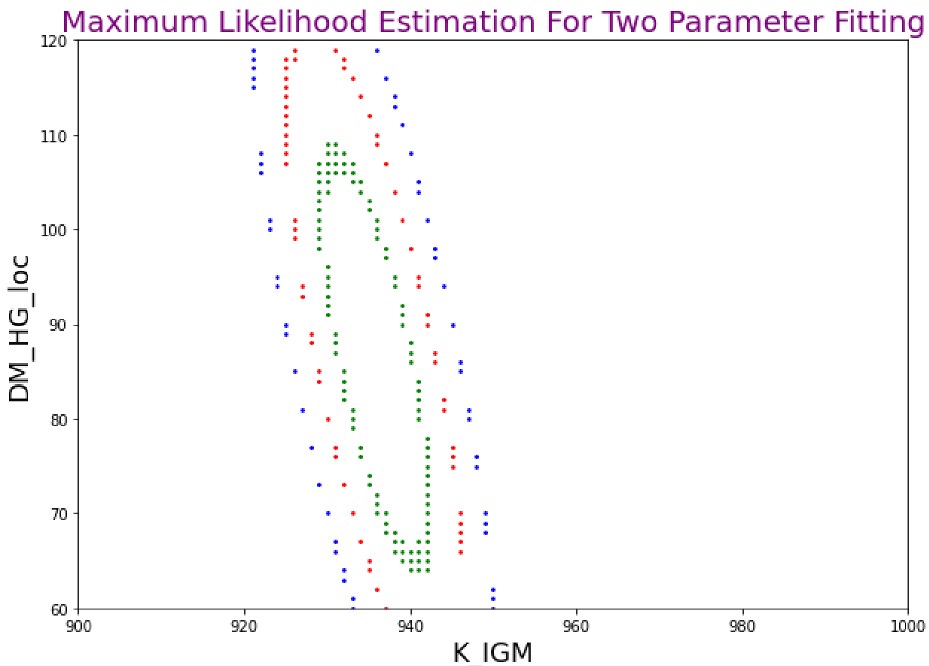

Fig.38

(Constraining $DM_{HG,loc}$ and K_{IGM}, we obtain a value of 574.432 as the chi-square minimum and the chi-square per degree of freedom is 1.148 for a $DM_{HG,loc}$ value of 86 pc cm^{-3} and K_{IGM} value of 936)

Similarly for 500 Samples and $DM_{HG,loc}$=N(100 pc cm^{-3}, 50 pc cm^{-3})
<u>(Appendix 3.1)</u>

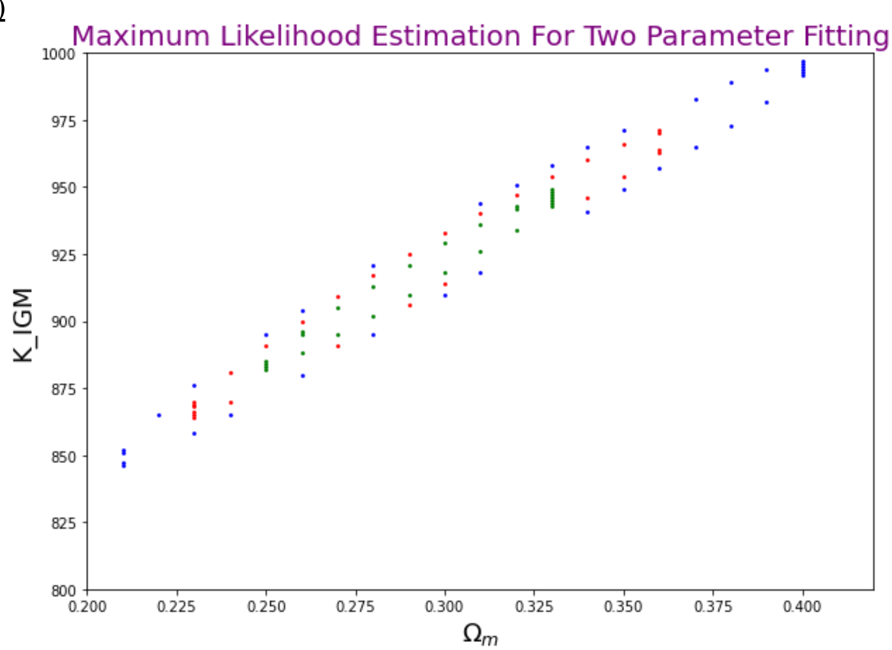

Fig.39

(Constraining K_{IGM} and Ω_m, we obtain a value of 447.181 as the chi-square minimum and the chi-square per degree of freedom is 0.89 for a K_{IGM} value of 916 and Ω_m value of 0.29)

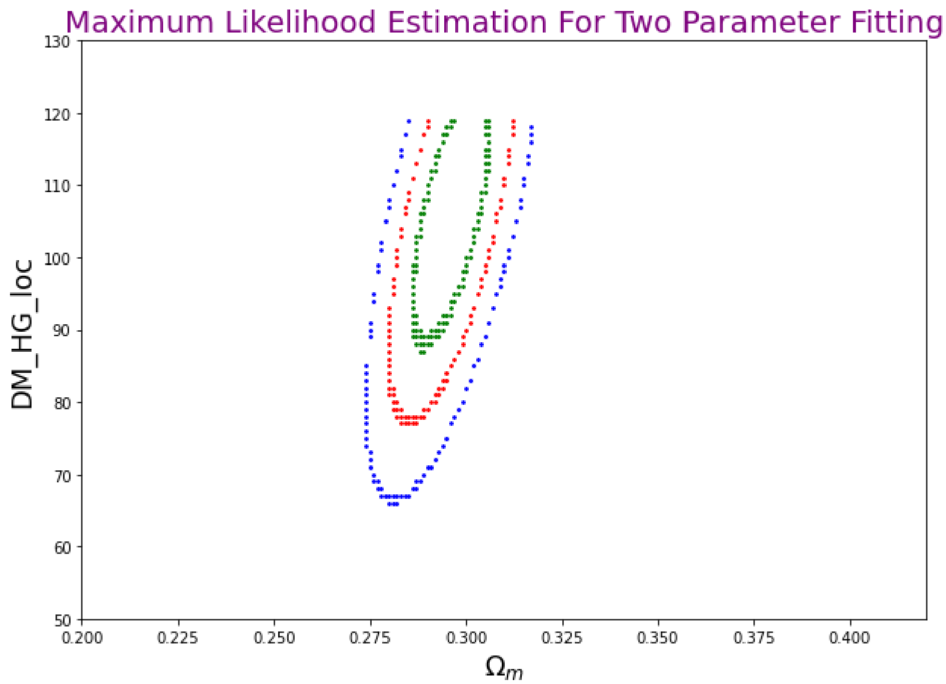

Fig.40

(Constraining $DM_{HG,loc}$ and Ω_m, we obtain a value of 447.181 as the chi-square minimum and the chi-square per degree of freedom is 0.89 for a $DM_{HG,loc}$ value of 106 pc cm^{-3} and Ω_m value of 0.296)

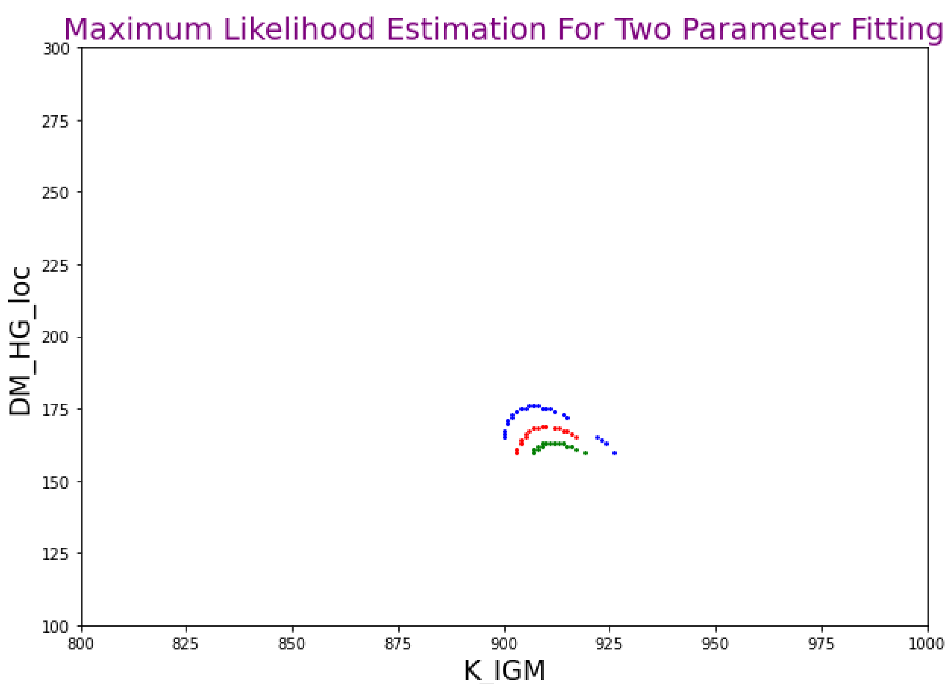

Fig.41

(Constraining $DM_{HG,loc}$ and K_{IGM}, we obtain a value of 467.275 as the chi-square minimum and the chi-square per degree of freedom is 0.93 for a $DM_{HG,loc}$ value of 160 pc cm^{-3} and K_{IGM} value of 913)

Similarly for 500 Samples and $DM_{HG,loc}$=N(200 pc cm^{-3}, 50 pc cm^{-3})
(Appendix 3.2)

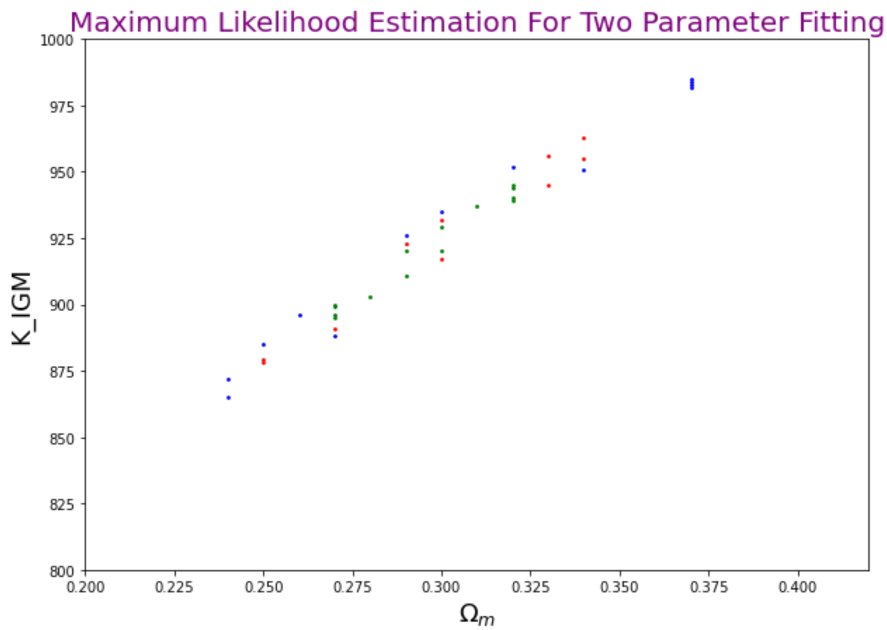

Fig.42

(Constraining K_{IGM} and Ω_m, we obtain a value of 552.651 as the chi-square minimum and the chi-square per degree of freedom is 1.105 for a K_{IGM} value of 916 and Ω_m value of 0.29)

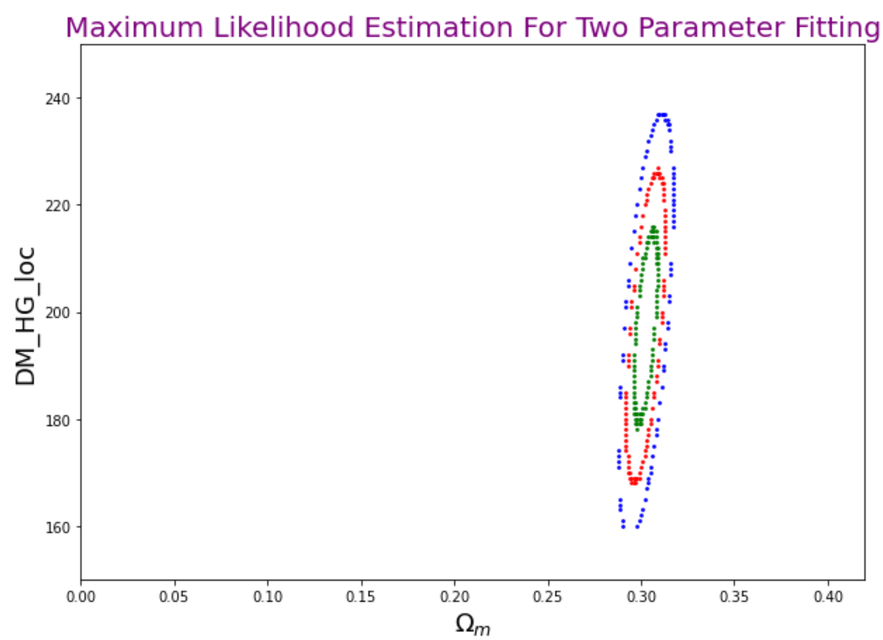

Fig.43

Constraining $DM_{HG,loc}$ and Ω_m, we obtain a value of 552.074 as the chi-square minimum and the chi-square per degree of freedom is 1.104 for a $DM_{HG,loc}$ value of 197 pc cm^{-3} and Ω_m value of 0.302.

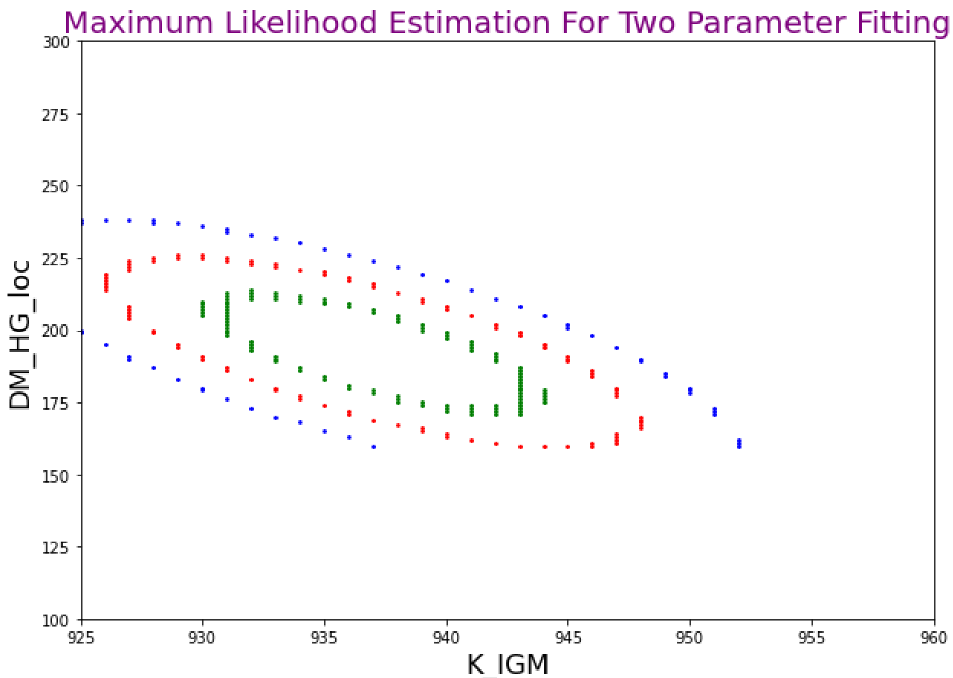

Fig.44

(Constraining $DM_{HG,loc}$ and K_{IGM}, we obtain a value of 551.896 as the chi-square minimum and the chi-square per degree of freedom is 1.103 for a $DM_{HG,loc}$ value of 192 pc cm^{-3} and K_{IGM} value of 937)

3.3 Conclusions

We generated 6 different mock data samples, 5 consisting of 500 data points and one consisting of 50 data points,by first changing the values of z_f (= 3, 2, 1) for a fixed normal distribution of $DM_{HG,loc}$ (100 pc cm^{-3}, 20 pc cm^{-3}) and then changing the normal distribution of $DM_{HG,loc}$, N(100 pc cm^{-3} , 50 pc cm^{-3}) and N (200 pc cm^{-3}, 50 pc cm^{-3}) with z_f= 3. After analyzing the log DM-log z plot, we saw that slight change in the distribution of z doesn't affect the global shape of the scatter plot.

Data Points	z_f	Normal Distribution of $DM_{HG,loc}$	Constrained Parameters	Value of constrained Parameters at Chi -square min.	Chi-Square minimum	Chi-Square per degree of freedom
500	3	(100 pc cm^{-3}, 20 pc cm^{-3})	K_{IGM}	917	574.056	1.148
			Ω_m	0.291		
			$DM_{HG,loc}$	85 pc cm^{-3}	574.466	1.148
			Ω_m	0.311		
			$DM_{HG,loc}$	86 pc cm^{-3}	574.432	1.148
			K_{IGM}	936		

Data Points	z_f	Normal Distribution of $DM_{HG,loc}$	Constrained Parameters	Value of constrained Parameters at Chi -square min.	Chi-Square minimum	Chi-Square per degree of freedom
500	2	(100 pc cm^{-3}, 20 pc cm^{-3})	K_{IGM}	965	527.885	1.050
			Ω_m	0.350		
			$DM_{HG,loc}$	109 pc cm^{-3}	527.365	1.054
			Ω_m	0.311		
			$DM_{HG,loc}$	109 pc cm^{-3}	527.370	1.054
			K_{IGM}	930		

Data Points	z_f	Normal Distribution of $DM_{HG,loc}$	Constrained Parameters	Value of constrained Parameters at Chi -square min.	Chi-Square minimum	Chi-Square per degree of freedom
500	1	(100 pc cm^{-3}, 20 pc cm^{-3})	K_{IGM}	1008	494.436	0.988
			Ω_m	0.480		
			$DM_{HG,loc}$	119 pc cm^{-3}	494.082	0.988
			Ω_m	0.341		
			$DM_{HG,loc}$	119 pc cm^{-3}	494.520	0.980
			K_{IGM}	917		

Data Points	z_f	Normal Distribution of $DM_{HG,loc}$	Constrained Parameters	Value of constrained Parameters at Chi -square min.	Chi-Square minimum	Chi-Square per degree of freedom
500	3	(100 pc cm^{-3}, 50 pc cm^{-3})	K_{IGM}	916	447.181	0.89
			Ω_m	0.290		
			$DM_{HG,loc}$	106 pc cm^{-3}	447.181	0.89
			Ω_m	0.296		
			$DM_{HG,loc}$	160 pc cm^{-3}	467.275	0.93
			K_{IGM}	913		

Data Points	z_f	Normal Distribution of $DM_{HG,loc}$	Constrained Parameters	Value of constrained Parameters at Chi -square min.	Chi-Square minimum	Chi-Square per degree of freedom
500	3	(200 pc cm^{-3}, 50 pc cm^{-3})	K_{IGM}	916	552.651	1.105
			Ω_m	0.290		
			$DM_{HG,loc}$	197 pc cm^{-3}	552.074	1.104
			Ω_m	0.302		
			$DM_{HG,loc}$	192 pc cm^{-3}	551.896	1.103
			K_{IGM}	937		

Data Points	z_f	Normal Distribution of $DM_{HG,loc}$	Constrained Parameters	Value of constrained Parameters at Chi -square min.	Chi-Square minimum	Chi-Square per degree of freedom
50	3	(100 pc cm^{-3}, 20 pc cm^{-3})	K_{IGM}	1049	56.888	1.137
			Ω_m	0.440		
			$DM_{HG,loc}$	131 pc cm^{-3}	55.239	1.100
			Ω_m	0.392		
			$DM_{HG,loc}$	139 pc cm^{-3}	55.057	1.101
			K_{IGM}	980		

For further study, we will try to use emcee to obtain the probability distribution of the fitting parameters.

REFERENCES:

Adekitan, A. I. 2014. “MONTE CARLO SIMULATION.” ResearchGate. http://dx.doi.org/10.13140/RG.2.2.15207.16806.

Berger, Roger L., and George Casella. 2002. *Statistical Inference*. Edited by Brooks/Cole Publishing Company. N.p.: Thomson Learning.

Chu, Jennifer. 2021. “CHIME Telescope Detects More Than 500 Mysterious Fast Radio Bursts From Outer Space.” SciTechDaily. https://scitechdaily.com/chime-telescope-detects-more-than-500-mysterious-fast-radio-bursts-from-outer-space/.

Devlin, Hannah. 2018. “Astronomers may be closing in on source of mysterious fast radio bursts.” The Guardian. https://www.theguardian.com/science/2018/jan/10/astronomers-may-be-closing-in-on-source-of-mysterious-fast-radio-bursts.

“Fast radio burst.” n.d. Wikipedia. Accessed July 3, 2022. https://en.wikipedia.org/wiki/Fast_radio_burst#Features.

Kroese, Dirk P. 2014. “Why the Monte Carlo method is so important today.” Semantic Scholar. https://www.semanticscholar.org/paper/Why-the-Monte-Carlo-method-is-so-important-today-Kroese-Brereton/7a56b632de84d0b81f283750b11609a042890639.

Lorimer. 2007. “A Bright Millisecond Radio Burst of Extragalactic Origin.” https://www.science.org/doi/10.1126/science.1147532.

Romero, Gustavo E. 2017. “AN OVERVIEW OF FAST RADIO BURSTS.” Instituto de Astronomía de la UNAM. http://www.astroscu.unam.mx/rmaa/RMxAC..49/PDF/RMxAC..49_reviews-2.pdf.

Wolpert, Robert L., and James O. Berger. 1988. *The Likelihood Principle*. N.p.: Institute of Mathematical Statistics.

https://prappleizer.github.io/Tutorials/MCMC/MCMC_Tutorial_Solution.html

https://machinelearningmastery.com/markov-chain-monte-carlo-for-probability/

APPENDIX

Appendix 1.1

```
#Code for find the value of pi by using Monte Carlo Simulation(Area of
Circle)
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import seaborn as sn
import math

#--------GETTING RANDOM VARIABLES----------
N=int(input("Enter the value of N = ")) #Total Number of turns
r=int(input("Radius of the circle = "))
c=0  #times when target hits inside the circle
x=np.random.uniform(0,r,N)  #Syntax: np.random.uniform(low,high,size)
y=np.random.uniform(0,r,N)
N1=np.linspace(1,N,N)
#-------------Empty List-----------------
f=[]
pi_o=[]

plt.figure(figsize=(20,7))
plt.subplot(1,2,1)      #Row 1 Column 2 Index 1
for i in range(N):
  if (x[i]**2+y[i]**2)<=r**2:
    c=c+1
    plt.scatter(x[i],y[i],color="red",marker=".",s=10)
  else:
    plt.scatter(x[i],y[i],color="blue",marker=".",s=10)
  pi=4*c/(i+1)
  pi_o.append(3.14)
  f.append(pi)
print("Value of pi using Monte Carlo Simulation = ",pi)

plt.xlabel('x-axis',size='20',color='brown')
plt.ylabel('y-axis',size='20',color='brown')
plt.title("Scattered plot",size='30',color='purple')
```

```
#plt.show()  #function to show the plot

plt.subplot(1,2,2)
#plt.figure(figsize=(10,7))
plt.plot(N1,f,color="brown",)
plt.plot(N1,pi_o,linestyle="dashed",color="green")
plt.xlim(0,N)
plt.xlabel("No. of turns",size='20',color='brown')
plt.ylabel("Pi",size='20',color='brown')
plt.title("Monte Carlo Simulation",size='30',color='purple')
plt.grid()
plt.show()
```

Appendix 1.2:

```
#Code for find the value of pi by using Monte Carlo Simulation(Volume of
Sphere)
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import seaborn as sn
import math

#-----Generate points------
N=int(input("Enter the value: "))
r=int(input("Radius of the circle = "))
x,y,z= np.random.uniform(-r,r,size=N), np.random.uniform(-r,r,size=N),
np.random.uniform(-r,r,size=N);
c=0      #Count for no. of times dart is hit inside the sphere

#-------------Empty List-----------------
N1=np.linspace(1,N,N)
f=[]
pi_o=[]

plt.figure(figsize=(10,10))
ax=plt.axes(projection='3d')  #Syntax for 3d projection

for i in range(N):
    if (x[i]**2 + y[i]**2 + z[i]**2) <=r**2 :
```

```
      c=c+1
      ax.scatter3D(x[i],y[i],z[i],alpha=0.4,marker='.',color='red')
    else:
      ax.scatter3D(x[i],y[i],z[i],alpha=0.4,marker='.',color='blue')
    pi=6*c/(i+1)
    f.append(pi)
    pi_o.append(3.14)
print("Value of pi using Monte Carlo Simulation = ",pi)

#-------PLOTTING---------

#Adding x and y gridlines
ax.grid(color ='grey',linestyle ='-.', linewidth = 0.3,alpha = 0.2)

plt.title("3D scatter plot",fontsize='25',color='purple')

ax.set_xlabel('X-axis', fontweight ='bold',fontsize='15',color='brown')
ax.set_ylabel('Y-axis', fontweight ='bold',fontsize='15',color='brown')
ax.set_zlabel('Z-axis', fontweight ='bold',fontsize='15',color='brown')
plt.show()

plt.figure(figsize=(7,7))
plt.plot(N1,f,color="brown",)
plt.plot(N1,pi_o,linestyle="dashed",color="green")
plt.xlim(0,N)
plt.xlabel("No. of turns",size='20',color='brown')
plt.ylabel("Pi",size='20',color='brown')
plt.title("Monte Carlo Simulation",size='30',color='purple')
plt.grid()
plt.show()
```

Appendix 1.3:

```
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import seaborn as sn
import math
```

```
#-------------------------WITHOUT TOLERANCE------------------------------

N=10000
#----------Integral between 0 to pi-----------------
x=np.random.uniform(0,2*np.pi,N)
y=np.random.uniform(-1,1,N)
y1=[]
x.sort()
c=0

plt.figure(figsize=(10,7))
for i in range(len(x)):
  y0=math.sin(x[i])
  if x[i]<=np.pi:
    if y[i]>=0 and y[i]<=y0:
      plt.scatter(x[i],y[i],s=5,alpha=0.6,c='cadetblue')
      c+=1
    else:
      plt.scatter(x[i],y[i],s=5,alpha=0.6,c='y')

  else:
    if y[i]<=0 and y[i]>=y0:
      plt.scatter(x[i],y[i],s=5,alpha=0.6,c='cadetblue')
      c+=1
    else:
      plt.scatter(x[i],y[i],s=5,alpha=0.6,c='y')

  y1.append(y0)

print('Total no. of darts inside the contour = ',c)
I=2*(2*np.pi-0)*c/N
print("\nIntegral with 3 decimal place accuracy: ",I)
plt.plot(x,y1,color='red')
plt.axhline(y=0,linestyle='--',linewidth=2.5,c='black')
plt.xlabel("x-axis",size='20',color='brown')
plt.ylabel("y-axis",size='20',color='brown')
plt.title("Contour to be Integrated",size='30',color='purple')
plt.grid()
plt.savefig("l1.png")
plt.show()
```

Appendix 1.4:

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
# Axes3D import has side effects, it enables using projection='3d' in
add_subplot

import matplotlib.pyplot as plt
import random

V=1*1*1 # Target Hyper volume
def fun(x, y):
    return y*x**2

N=10000
fig = plt.figure(figsize=(15,9))
ax = fig.add_subplot(111, projection='3d')
x = np.random.uniform(0,1,N)
y = np.random.uniform(0,1,N)
z = np.random.uniform(0,1,N)
Zf = fun(x,y)

ax.scatter3D(x, y, z, color = "yellow", marker='.')

X, Y = np.meshgrid(x, y)
tmp = np.array(fun(np.ravel(X), np.ravel(Y)))
Z = tmp.reshape(X.shape)
plt.title("Contour to be Integrated",size='30',color='purple')
ax.plot_surface(X, Y, Z)

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
plt.show()

#------
c1=0
for i,j in zip(z,Zf):
    if i<=j:
```

```
        c1+=1

I=V*c1/N
print('\nIntegral= ',I)
plt.savefig("l2.png")
```

Appendix 1.5:

```
#----Importing Libraries----
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style("darkgrid", {"axes.facecolor": ".8"}) # Setting Background
#To find the cost of Pudding using Monte Carlo Simulation
import pandas as pd
N=1
mean=np.array([])

while(N<=1000):
  cost=np.array([])
  milk= np.random.uniform(18,20,N)          #for one lt
  sugar= np.random.uniform(20,23,N)          #for one kg
  bread = np.random.normal(loc=25,scale=1,size=N)  #for 12 slices
  egg= np.random.choice([2,2.5,3],p=[0.2,0.5,0.3],size=N) #For 1 egg
  almond= np.random.choice([500,700],p=[0.7,0.3],size=N) #For 0.5 kg

  for i in range(0,N):
    cost_per=(milk[i] + 2*egg[i] + 0.1*sugar[i] + (1/6)*bread[i] +
0.1*almond[i])
    cost=np.append(cost,cost_per)

  #print('For value of N:',N)
  mean_value=sum(cost)/N     #To find the mean cost for particular data
points generated
  mean=np.append(mean,mean_value) #Append the mean_value to mean ndarray
  #print('Mean of cost of pudding:',mean_value)
  #plt.hist(cost,bins=100)    #To plot the histogram
  #plt.show()

  N+=1    #To increment the number of points generated
```

```
#plt.hist(cost,bins=100)   #To plot the histogram
  #plt.show()

d=np.array([cost,almond,milk,sugar,bread,egg]).T
#print(d)
print('Mean of cost of pudding:',mean_value,"\n")
data=pd.DataFrame(data=d,
columns=["cost","almond","milk","sugar","bread","egg"])
plt.hist(cost,bins=100)   #To plot the histogram
plt.show()

# Correlogram
plt.figure(figsize=(10,7))
sns.heatmap(data.corr(),cmap="YlGnBu", annot=True)
plt.show()
```

Appendix 1.6:

Generate Sample data for the following probability density function using MCMC

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

# Probability Density Function
def targetdist(x):
  probX = abs(np.cos(x**2)*(21+np.sin(x+3)))/(np.exp(x**2)+np.exp(-x))
  return probX
x = np.arange(-8,8, 0.01)
y = targetdist(x)

plt.figure(figsize=(9,7))
plt.plot(x, y)
plt.title("Probability Density Function")
plt.show()

# Markov chain Method with only 1
walker----------------------------------------------

nchain=[]
achain=[]
post_chain=[]          # Posterior Chain
```

```
a0=-5                   # Initial value
s=50000                 # No of steps
burnout_time= 200       # Burnout Time
t=0

for i in range(1,s):
  a_new= a0+np.random.normal(0,0.2)
  alpha = targetdist(a_new) / targetdist(a0)
  u=np.random.uniform(0,1)

  if alpha>=u:
    t=t+1
    if t>=burnout_time:
      nchain.append(t)
      achain.append(a_new)
      post_chain.append(targetdist(a_new))
    a0=a_new
  else:
    a0=a0

# Plot Markov chain
plt.plot(nchain,achain)
plt.title("Plot Markov chain")
plt.show()

"""
def targetdist(x):
  probX = abs(np.sin(x+2)/(1+x**2))
  return probX
x = np.arange(-8,8, 0.01)
y = targetdist(x)
"""

plt.figure(figsize=(9,7))
plt.plot(x,y)
plt.title("Plot the chain over the distribution function")

for i in range(0,10000):
  plt.scatter(achain[i],post_chain[i],color='red')
plt.show()
```

```
sl,mean,sm= np.percentile(achain,[16, 50, 84],axis=0)
neg_error= mean-sl
pos_error= sm-mean
print("MCMC OUTCOME:= {} + {} -{}".format(mean,pos_error,neg_error))
meas=[mean,-neg_error,pos_error]
name=["Mean","+ Error","- Error"]

plt.figure(figsize=(9,7))
plt.hist(achain, bins=40, density=True, alpha=0.6, color='cyan',
edgecolor='black')
for measurement, name, color in zip(meas,name,["red","green","green"]):
    plt.axvline(x=measurement, linestyle='--', linewidth=2.5, label='{0}
at {1}'.format(name,round(measurement,3)), c=color)
plt.xlabel("Data Points",size='12')
plt.ylabel("Probability Density",size='12')
plt.legend(loc="upper right")
plt.show()

import numpy as np
import matplotlib.pyplot as plt
import corner
# Markov chain Method with multiple walkers---------------------------

walker= 10
nchain_list = []
achain_list = []
post_chain_list=[]

for i in range(0,walker):
  t=0
  a0=np.random.normal(0,1)
  steps=2000
  burnout_time= 100
  nchain=[]
  achain=[]
  posteriorchain=[]

  for j in range(0,steps):
    anew= a0+np.random.normal(0,0.1)
    alpha = targetdist(anew) / targetdist(a0)
```

```
    u=np.random.uniform(0,1)

    if alpha>=u:
      t=t+1
      if t>=burnout_time:
        nchain.append(t)
        achain.append(anew)
        posteriorchain.append(targetdist(anew))
      a0=anew

  nchain_list.append(nchain)
  achain_list.append(achain)
  post_chain_list.append(posteriorchain)
merge=[] # Container for conataining every value of achain in simple 1-D
array

plt.figure(figsize=(9,7))
for i in range(0,walker):
  n=nchain_list[i]
  a=achain_list[i]
  merge.extend(a)
  plt.plot(n,a)

plt.show()

sl,mean,sm= np.percentile(merge,[16, 50, 84],axis=0)
neg_error= mean-sl
pos_error= sm-mean
print("MCMC OUTCOME:= {} + {} -{}".format(mean,pos_error,neg_error))
meas=[mean,-neg_error,pos_error]
name=["Mean","+ Error","- Error"]

plt.figure(figsize=(9,7))
plt.hist(merge, bins=40, density=True, alpha=0.6, color='orange',
edgecolor='black')
for measurement, name, color in zip(meas,name,["red","green","green"]):
    plt.axvline(x=measurement, linestyle='--', linewidth=2.5, label='{0}
at {1}'.format(name,round(measurement,3)), c=color)
plt.xlabel("Data Points",size='12')
plt.ylabel("Probability Density",size='12')
```

```
plt.legend(loc="upper right")
plt.show()
```

Appendix 1.7:

Maximum Likelihood Estimation for omega matter (Omega_M) (ONE PARAMETER FITTING)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math as m

# Dataset
#d=pd.read_excel("/content/Likelihood_1.xlsx")
d=pd.read_excel("/content/Likelihood_2.xlsx")
d

def Hubble(z,omg_M):
  H0=67.8
  Hth=[]
  for i in z:
    t1= H0*m.sqrt((omg_M*pow((1+i),3)+(1-omg_M)))
    Hth.append(t1)
  return Hth
X2=[]
omg_M=[]
d_new=d.copy(deep=True)

for o_m in np.arange(0,0.8,0.0005):
  omg_M.append(o_m)

  H_th=np.array(Hubble(d["z"].to_numpy(),o_m))

  d_new["H_th"]=H_th
  #print(d_new)

  t2=0
  for i,j,k in zip(d_new["H(z)"],d_new["H_th"],d_new["Sigma"]):
    t2+=pow(i-j,2)/pow(k,2)
```

```
  X2.append(t2)

# Coverting to dataframe to ease further calcuations
d1=np.array([omg_M,X2]).T
df=pd.DataFrame(data=d1, columns=["omg_M","X2"])
print(df)

# Conclusion
X2_min=df['X2'].min()
Omega_M=df.loc[df['X2'] == X2_min,'omg_M'].values[0]

print("Chi Square is minimum at: ", X2_min)
print("Value of Omega matter at chi Square Minimum: ", Omega_M)

# Finding Standard Deviation using t-score method
def variance(data):
  n = len(data)
  mean = sum(data) / n
  return sum((x - mean) ** 2 for x in data) / n

def stdev(data):
  var = variance(data)
  std_dev = m.sqrt(var)
  return std_dev

s=stdev(df["omg_M"])
sigma_1=[Omega_M+s,Omega_M-s]
sigma_2=[Omega_M+2*s,Omega_M-2*s]  #2*sigma_1
print(sigma_1)
print(sigma_2)

# Plot
plt.figure(figsize=(10,7))
plt.plot(df["omg_M"],df["X2"])
plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel(r'$\chi^{2}$',size='18')
plt.title('Maximum Likelihood Estimation For One Parameter
Fitting',color='purple',size='20')

name=[r'+$\sigma$',r'- $\sigma$']
```

```
plt.axvline(x=Omega_M, linestyle='--', linewidth=2.5, label='Most Likely Omega_M is {0}'.format(round(Omega_M,3)), c='red')
for measurement, name, color in zip(sigma_1,name,["green","green"]):
    plt.axvline(x=measurement, linestyle='--', linewidth=2.5, label='{0} at {1}'.format(name,round(measurement,3)), c=color)

plt.legend(loc="upper right")
plt.savefig("like-1.png")
plt.show()
```

Appendix 1.8:

Maximum Likelihood Estimation (TWO PARAMETER FITTING)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math as m
# Dataset
d=pd.read_excel("/content/Likelihood_2.xlsx")
d
def Hubble(z,omg_M,omg_L):
  H0=67.8
  Hth=[]
  for i in z:
    t1= H0*m.sqrt((omg_M*pow((1+i),3)+omg_L))
    Hth.append(t1)
  return Hth
#----------------FINAL MEASURE---------------------------------

X2_min=[]
omg_M=[]
omg_L=[]
X2=[]
d_1=d.copy(deep=True)

for o_m in np.arange(0,1,0.0006):
  omg_M.append(o_m)

  omg_L_tmp=[]
  X2_sub=[]
```

```
  for o_l in np.arange(0,1,0.0006):
    omg_L_tmp.append(o_l)
    H_th=np.array(Hubble(d["z"].to_numpy(),o_m,o_l))

    t2=0
    t3=0
    for i,j,k in zip(d_1["H(z)"],H_th,d_1["Sigma"]):
      t2=t2+pow((i-j),2)/pow(k,2)
    X2_sub.append(t2)
    t3=min(X2_sub)
  X2.append(X2_sub)
  omg_L.append(omg_L_tmp)
  X2_min.append(t3)

# Coverting to dataframe to ease further calcuations
d1=np.array([omg_M,omg_L,X2,X2_min]).T
d_f=pd.DataFrame(data=d1, columns=["omg_M","omg_L","X2","X2_min"])
d_f
Min=min(d_f["X2_min"])
id_1=d_f["X2_min"].astype('float64').idxmin() # Index of min(X2_min)

id_2=d_f["X2"][id_1].index(min(d_f["X2"][id_1]))
print(Min,id_1,id_2)
#For 1-Sigma
x1=[]
y1=[]
#For 2-Sigma
x2=[]
y2=[]
#For 3-Sigma
x3=[]
y3=[]
for i in range(0,len(d_f["omg_M"])):
  for j in range(0,len(d_f["omg_L"][0])):
    t=d_f["X2"][i]
    if (t[j] < Min+2.3+0.05) and (t[j]> Min+2.3-0.05):  # 1-Sigma
      x1.append(d_f["omg_M"][i])
      y1.append(d_f["omg_L"][i][j])
    if (t[j] < Min+6.17+0.05) and (t[j]> Min+6.17-0.05): # 2-Sigma
      x2.append(d_f["omg_M"][i])
```

```
      y2.append(d_f["omg_L"][i][j])
    if (t[j] < Min+11+0.05) and (t[j]> Min+11-0.05): # 3-Sigma
      x3.append(d_f["omg_M"][i])
      y3.append(d_f["omg_L"][i][j])

# Plot
plt.figure(figsize=(10,7))
plt.scatter(d_f["omg_M"][id_1],d_f["omg_L"][id_1][id_2],s=6)
plt.scatter(x1,y1,color="green",s=0.6)
plt.scatter(x2,y2,color="crimson",s=0.6)
plt.scatter(x3,y3,color="navy",s=0.6)
plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel(r'$\Omega_L$',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.savefig("like-1.png")
plt.show()
```

Appendix 1.9:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import seaborn as sn
c=2.99792458*10**(10)              # cm/s
H0=67.7*(10**(-1))        #cm s^(-1) pc(-1)
omg_b=0.049
f_IGM=0.83
pi=3.14
G=6.67*(10**(-8))      #cm^3 g^(-1) s^(-2)
m_p=1.67*10**(-24)     # g

#Scaling Constant
K_IGM=(3*c*H0*omg_b*f_IGM)/(8*pi*G*m_p)
K_IGM=K_IGM/(3.08*3.08*10**(36))
print("Scaling factor = ",K_IGM)       #Units= pc/cm^3

#-----DM_IGM------
import numpy as np
from scipy.integrate import trapz
Omg_m=[0.1,0.3,0.5]
```

```
f_e=7/8
DM_IGM=[]
z=np.arange(0,3.0,0.001)

for omg_m in Omg_m:
  tmp=[]
  for i in z:
    x = np.linspace(0, i, 100)
    fz=(K_IGM*f_e*(1+x))/np.sqrt(((omg_m*(1+x)**3)+(1-omg_m)))
    I_trapz = trapz(fz,x)
    tmp.append(I_trapz)
  DM_IGM.append(tmp)

#----Plot for relation between <DM_IGM>-z------
fig, ax = plt.subplots(figsize=(10,7))
ax.plot(z,DM_IGM[0],label="Omg_m=0.1")
ax.plot(z,DM_IGM[1],label="Omg_m=0.3")
ax.plot(z,DM_IGM[2],label="Omg_m=0.5")
plt.ylabel("<DM_IGM>",size='18')
plt.xlabel('z',size='18')
plt.title('<DM_IGM>-z Relation',color='purple',size='20')
plt.legend()
plt.ylim([0,3000])
plt.show()

#-----alpha------
from scipy.integrate import trapz

Y_=[]
z=np.arange(0,3,0.01)
OMG_M=[0.1,0.3,0.5]

for omg_M in OMG_M:
  tmp=[]
  for i in z:
    nu=(i*(7/8)*(1+i))/np.sqrt(omg_M*pow(1+i,3)+(1-omg_M))
    x = np.linspace(0, i, 100)
    f=((7/8)*(1+x))/np.sqrt(omg_M*pow(1+x,3)+(1-omg_M))
    I_trapz = trapz(f,x)
    tmp.append(nu/I_trapz)
```

```
  Y_.append(tmp)
#-----Plot of alpha-z relation------
fig, ax = plt.subplots(figsize=(10,7))
ax.plot(z,Y_[0],label="Omg_m=0.1")
ax.plot(z,Y_[1],label="Omg_m=0.3")
ax.plot(z,Y_[2],label="Omg_m=0.5")
plt.xlabel('z',size='18')
plt.ylabel(r'$\alpha$',size='18')
plt.title(r'$\alpha$-z Relation',color='purple',size='20')
plt.legend()
plt.xlim()
plt.show()
```

<u>Appendix 2.0:</u>

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import seaborn as sn
c=2.99792458*10**(10)                # cm/s
H0=67.7*(10**(-1))        #cm s^(-1) pc(-1)
omg_b=0.049
f_IGM=0.83
pi=3.14
G=6.67*(10**(-8))       #cm^3 g^(-1) s^(-2)
m_p=1.67*10**(-24)     # g

#Scaling Constant
K_IGM=(3*c*H0*omg_b*f_IGM)/(8*pi*G*m_p)
K_IGM=K_IGM/(3.08*3.08*10**(36))
print("Scaling factor = ",K_IGM)       #Units= pc/cm^3
#-----<DM_E>-------
import numpy as np
from scipy.integrate import trapz

omg_m=0.31
f_e=7/8
DM_IGM=[]
z=np.arange(0,3.0,0.01)

for i in z:
```

```
  x = np.linspace(0, i, 100)
  fz=(K_IGM*f_e*(1+x))/np.sqrt(((omg_m*(1+x)**3)+(1-omg_m)))
  I_trapz = trapz(fz,x)
  DM_IGM.append(I_trapz)
DM_Hos_Loc=[0,60,120]
z=np.arange(0,3.0,0.01)
DM_E=[]

for d in DM_Hos_Loc:
  tmp=[]
  for j,k in zip(DM_IGM,z):
    t1=j+(d/(1+k))
    tmp.append(t1)
  DM_E.append(tmp)
#----Plot of <DM_E>-z relation------
fig, ax = plt.subplots(figsize=(10,7))
ax.plot(z,DM_E[0],label="DM_Hos_Loc=0 pc cm^-3")
ax.plot(z,DM_E[1],label="DM_Hos_Loc=60 pc cm^-3")
ax.plot(z,DM_E[2],label="DM_Hos_Loc=120 pc cm^-3")
plt.legend()
plt.ylabel("<DM_E>",size='18')
plt.title('<DM_E>-z Relation',color='purple',size='20')
plt.xlim([0.1,3])
plt.ylim([0,2000])
ax.yaxis.set_ticks(np. arange(0,2000,100))
plt.show()
#---- Beta-----
from scipy.integrate import trapz

Y_=[]
z=np.arange(0,3,0.01)
omg_M=0.31

for i in z:
  nu=(i*(7/8)*(1+i))/np.sqrt(omg_M*pow(1+i,3)+(1-omg_M))
  x = np.linspace(0, i, 100)
  f=((7/8)*(1+x))/np.sqrt(omg_M*pow(1+x,3)+(1-omg_M))
  I_trapz = trapz(f,x)
  Y_.append(nu/I_trapz)
DM_Hos_Loc=[0,60,120]
```

```
B_=[]
for d1,e in zip(DM_Hos_Loc,range(len(DM_E))):
  t=[]
  for m,n,o in zip(z,DM_E[e], Y_):
    b=((n-(d1/(1+m)))*o/n)-(d1*m/(n*pow(1+m,2)))
    t.append(b)
  B_.append(t)
#-----Plot of Beta-z relation------
plt.figure(figsize=(10,7))
plt.plot(z,B_[0],label="DM_Hos_Loc=0 pc cm^-3")
plt.plot(z,B_[1],label="DM_Hos_Loc=60 pc cm^-3")
plt.plot(z,B_[2],label="DM_Hos_Loc=120 pc cm^-3")
plt.xlabel('z',size='18')
plt.ylabel(r'$\beta$',size='18')
plt.title(r'$\beta$-z Relation',color='purple',size='20')
plt.legend()
plt.xlim()
plt.show()
```

Appendix 2.1:

```
pip install pynverse
import numpy as np
from scipy import integrate
import math as m
from pynverse import inversefunc
import pandas as pd
import matplotlib.pyplot as plt

#Constants
H_0=67.7   #Hubble Constant
omega_m=0.31
omega_l =0.69   #fraction of the energy of the universe due to the
cosmological constant
omega_b=0.049   #current baryon mass density fraction of the universe
f_IGM= 0.83     #fraction of baryon mass in IGM
K_IGM=933
m_p=1.67*(10**-27)
#P(z)=z*exp(-z)
PDF= lambda z : z* m.exp(-z)
CDF= lambda z: 1-m.exp(-z)*(z+1)
```

```
U=np.random.uniform(0,0.80,50)

z=[]
for i in range(len(U)):
  inv_CDF = inversefunc(CDF,U[i])
  z=np.append(z,inv_CDF)

z.sort()
z

DM_IGM=[]
DM_HG_loc=[]
DM_E=[]
function=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+0.69)**0.5
for i in range(0,50):
  Integration=integrate.quad(function,0,z[i])
  DM_IGM_f=K_IGM * Integration[0]
  DM_IGM=np.append(DM_IGM,np.random.normal(DM_IGM_f,100))
  DM_HG_loc=np.append(DM_HG_loc,np.random.normal(100,20))
  DM_E_formula=DM_IGM[i]+(DM_HG_loc[i]/(1+z[i]))
  DM_E=np.append(DM_E,DM_E_formula)

#print(DM_IGM)
print(DM_HG_loc)
print(DM_E)
poly_deg = 2
p = np.polyfit(log_z,log_DM_E, poly_deg)
print(p)
y_fit = np.poly1d(p) #use np.poly1d to give the polynomial function
corresponding to the coeffs. in p

#Plot the data and the fitted function:

plt.plot(log_z, y_fit(log_z), ls='-', label='polynomial of deg.
{}'.format(poly_deg),color='red')

plt.scatter(log_z,log_DM_E,s=4)
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
```

```
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("50 Samples(z_f=3).jpg")
```

Appendix 2.2:

```
pip install pynverse
import numpy as np
from scipy import integrate
import math as m
from pynverse import inversefunc
import pandas as pd
import matplotlib.pyplot as plt
#Constants
H_0=67.7   #Hubble Constant
omega_m=0.31
omega_l =0.69   #fraction of the energy of the universe due to the
cosmological constant
omega_b=0.049   #current baryon mass density fraction of the universe
f_IGM= 0.83     #fraction of baryon mass in IGM
K_IGM=933
m_p=1.67*(10**-27)
#P(z)=z*exp(-z)
PDF= lambda z : z* m.exp(-z)
CDF= lambda z: 1-m.exp(-z)*(z+1)

U=np.random.uniform(0,0.8,500)

z=[]
for i in range(len(U)):
  inv_CDF = inversefunc(CDF,U[i])
  z=np.append(z,inv_CDF)

z.sort()
DM_IGM=[]
DM_HG_loc=[]
DM_E=[]
function=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+0.69)**0.5
for i in range(0,500):
  Integration=integrate.quad(function,0,z[i])
  DM_IGM_f=K_IGM * Integration[0]
```

```
  DM_IGM=np.append(DM_IGM,np.random.normal(DM_IGM_f,100))
  DM_HG_loc=np.append(DM_HG_loc,np.random.normal(100,20))
  DM_E_formula=DM_IGM[i]+(DM_HG_loc[i]/(1+z[i]))
  DM_E=np.append(DM_E,DM_E_formula)
log_DM_E=np.log10(DM_E)
log_z=np.log10(z)
poly_deg = 2
p = np.polyfit(log_z,log_DM_E, poly_deg)
print(p)
y_fit = np.poly1d(p) #use np.poly1d to give the polynomial function
corresponding to the coeffs. in p

#Plot the data and the fitted function:

plt.plot(log_z, y_fit(log_z), ls='-', label='polynomial of deg.
{}'.format(poly_deg),color='red')

plt.scatter(log_z,log_DM_E,s=4)
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples(z_f=3).jpg")
```

Appendix 2.3:

```
pip install pynverse
import numpy as np
from scipy import integrate
import math as m
from pynverse import inversefunc
import pandas as pd
import matplotlib.pyplot as plt
#Constants
H_0=67.7   #Hubble Constant
omega_m=0.31
omega_l =0.69   #fraction of the energy of the universe due to the
cosmological constant
omega_b=0.049   #current baryon mass density fraction of the universe
f_IGM= 0.83     #fraction of baryon mass in IGM
```

```
K_IGM=933
m_p=1.67*(10**-27)
#P(z)=z*exp(-z)
PDF= lambda z : z* m.exp(-z)
CDF= lambda z: 1-m.exp(-z)*(z+1)

U=np.random.uniform(0,0.594,500)

z=[]
for i in range(len(U)):
  inv_CDF = inversefunc(CDF,U[i])
  z=np.append(z,inv_CDF)
z.sort()
DM_IGM=[]
DM_HG_loc=[]
DM_E=[]
function=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+0.69)**0.5
for i in range(0,500):
  Integration=integrate.quad(function,0,z[i])
  DM_IGM_f=K_IGM * Integration[0]
  DM_IGM=np.append(DM_IGM,np.random.normal(DM_IGM_f,100))
  DM_HG_loc=np.append(DM_HG_loc,np.random.normal(100,20))
  DM_E_formula=DM_IGM[i]+(DM_HG_loc[i]/(1+z[i]))
  DM_E=np.append(DM_E,DM_E_formula)
log_DM_E=np.log10(DM_E)
log_z=np.log10(z)
poly_deg = 2
p = np.polyfit(log_z,log_DM_E, poly_deg)
print(p)
y_fit = np.poly1d(p) #use np.poly1d to give the polynomial function
corresponding to the coeffs. in p

#Plot the data and the fitted function:
plt.plot(log_z, y_fit(log_z), ls='-', label='polynomial of deg.
{}'.format(poly_deg),color='red')
plt.scatter(log_z,log_DM_E,s=6)
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
```

```
plt.savefig("500 Samples(z_f=2).jpg")
```

Appendix 2.4:

```
pip install pynverse
import numpy as np
from scipy import integrate
import math as m
from pynverse import inversefunc
import pandas as pd
import matplotlib.pyplot as plt

#Constants
H_0=67.7   #Hubble Constant
omega_m=0.31
omega_l =0.69   #fraction of the energy of the universe due to the
cosmological constant
omega_b=0.049   #current baryon mass density fraction of the universe
f_IGM= 0.83     #fraction of baryon mass in IGM
K_IGM=933
m_p=1.67*(10**-27)
#P(z)=z*exp(-z)
PDF= lambda z : z* m.exp(-z)
CDF= lambda z: 1-m.exp(-z)*(z+1)

U=np.random.uniform(0,0.265,500)
z=[]
for i in range(len(U)):
  inv_CDF = inversefunc(CDF,U[i])
  z=np.append(z,inv_CDF)
z.sort()
DM_IGM=[]
DM_HG_loc=[]
DM_E=[]
function=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+0.69)**0.5
for i in range(0,500):
  Integration=integrate.quad(function,0,z[i])
  DM_IGM_f=K_IGM * Integration[0]
  DM_IGM=np.append(DM_IGM,np.random.normal(DM_IGM_f,100))
  DM_HG_loc=np.append(DM_HG_loc,np.random.normal(100,20))
  DM_E_formula=DM_IGM[i]+(DM_HG_loc[i]/(1+z[i]))
```

```
  DM_E=np.append(DM_E,DM_E_formula)
log_DM_E=np.log10(DM_E)
log_z=np.log10(z)
poly_deg = 2
p = np.polyfit(log_z,log_DM_E, poly_deg)
print(p)
y_fit = np.poly1d(p) #use np.poly1d to give the polynomial function 
corresponding to the coeffs. in p

#Plot the data and the fitted function:
plt.plot(log_z, y_fit(log_z), ls='-', label='polynomial of deg. 
{}'.format(poly_deg),color='red')
plt.ylim(1.6,3.2)
plt.scatter(log_z,log_DM_E,s=6)
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples(z_f=1).jpg")
```

Appendix 2.5:

```
#importing the necessary libraries
import numpy as np
from scipy import integrate
import math as m
from pynverse import inversefunc
import pandas as pd
import matplotlib.pyplot as plt

#Constants
H_0=67.7   #Hubble Constant
omega_m=0.31
omega_l =0.69   #fraction of the energy of the universe due to the 
cosmological constant
omega_b=0.049   #current baryon mass density fraction of the universe
f_IGM= 0.83     #fraction of baryon mass in IGM
K_IGM=933
m_p=1.67*(10**-27)

#P(z)=z*exp(-z)     Probabilty Distribution Function for redshift z
```

```
PDF= lambda z : z* m.exp(-z)
CDF= lambda z: 1-m.exp(-z)*(z+1)  #Cumulative Distribution Function

U=np.random.uniform(0,0.594,500)  #Generate 500 data points between 0 and 
0.8 uniformly

z=[]
for i in range(len(U)):
  inv_CDF = inversefunc(CDF,U[i])     # Calculate the inverse function
  z=np.append(z,inv_CDF)              # Append the values of z in array z[]

z.sort()    #to sort the z array

#Create three empty array DM_IGM,DM_HG_loc and DM_E for storing the values 
of DM_IGM,DM_HG_loc and DM_E
DM_IGM=[]
DM_HG_loc=[]
DM_E=[]
function=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+0.69)**0.5
for i in range(0,500):
  Integration=integrate.quad(function,0,z[i])
#integrate.quad function , integrate the given funnction, func from 0 to z
  DM_IGM_f=K_IGM * Integration[0]
#formula to find the value of DM_IGM,integration[0] stores the result of 
integration of function , func
  DM_IGM=np.append(DM_IGM,np.random.normal(DM_IGM_f,100))

#introduce a randomness by taking normal distribution N(DM_IGM,100 pc 
cm^(-3)) in DM_IGM
  DM_HG_loc=np.append(DM_HG_loc,np.random.normal(100,50))
#take DM_HG_loc as normal distribution N(100 pc cm^(-3),20 pc cm^(-3))
  DM_E_formula=DM_IGM[i]+(DM_HG_loc[i]/(1+z[i]))
#formula to calculate value of DM_E
  DM_E=np.append(DM_E,DM_E_formula)

log_DM_E=np.log10(DM_E)
# np.log10 calculate the log of all the data points in DM_E array and new 
array created is stored in log_DM_E
log_z=np.log10(z)
```

```
# np.log10 calculate the log of all the data points in z array and new
array created is stored in log_z

plt.scatter(log_z,log_DM_E,s=4)
#Scatter plot between datapoints of log_DM_E and log_z array
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples (DM_hg_loc=(100,50)).jpg")

poly_deg = 2
p = np.polyfit(log_z,log_DM_E, poly_deg)
print(p)
y_fit = np.poly1d(p)
#use np.poly1d to give the polynomial function corresponding to the
coeffs. in p

#Plot the data and the fitted function:
plt.plot(log_z, y_fit(log_z), ls='-', label='polynomial of deg.
{}'.format(poly_deg),color='red')  #to plot the best fit line of degree 2

plt.scatter(log_z,log_DM_E,s=4)
# to plot scatter plot between log_z and log_DM_E
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples (DM_hg_loc=(100,50)).jpg")
```

Appendix 2.6:

```
#importing the necessary libraries
import numpy as np
from scipy import integrate
import math as m
from pynverse import inversefunc
import pandas as pd
import matplotlib.pyplot as plt

#Constants
```

```
H_0=67.7   #Hubble Constant
omega_m=0.31
omega_l =0.69   #fraction of the energy of the universe due to the
cosmological constant
omega_b=0.049   #current baryon mass density fraction of the universe
f_IGM= 0.83     #fraction of baryon mass in IGM
K_IGM=933
m_p=1.67*(10**-27)

#P(z)=z*exp(-z)     Probabilty Distribution Function for redshift z
PDF= lambda z : z* m.exp(-z)
CDF= lambda z: 1-m.exp(-z)*(z+1) #Cumulative Distribution Function

U=np.random.uniform(0,0.801,500) #Generate 500 data points between 0 and
0.8 uniformly

z=[]
for i in range(len(U)):
  inv_CDF = inversefunc(CDF,U[i])   #Calculate the inverse function
  z=np.append(z,inv_CDF)            #Append the values of z in array z[]

z.sort()    #to sort the z array

# Create three empty array DM_IGM,DM_HG_loc and DM_E for storing the
values of DM_IGM,DM_HG_loc and DM_E
DM_IGM=[]
DM_HG_loc=[]
DM_E=[]
function=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+0.69)**0.5
for i in range(0,500):
  Integration=integrate.quad(function,0,z[i])
#integrate.quad function , integrate the given funnction, func from 0 to z
  DM_IGM_f=K_IGM * Integration[0]
#formula to find the value of DM_IGM,integration[0] stores the result of
integration of function , func
  DM_IGM=np.append(DM_IGM,np.random.normal(DM_IGM_f,100))
#introduce a randomness by taking normal distribution N(DM_IGM,100 pc
cm^(-3)) in DM_IGM
  DM_HG_loc=np.append(DM_HG_loc,np.random.normal(200,50))
#take DM_HG_loc as normal distribution N(100 pc cm^(-3),20 pc cm^(-3))
```

```
  DM_E_formula=DM_IGM[i]+(DM_HG_loc[i]/(1+z[i]))
#formula to calculate value of DM_E
  DM_E=np.append(DM_E,DM_E_formula)

log_DM_E=np.log10(DM_E)      #np.log10 calculate the log of all the data
points in DM_E array and new array created is stored in log_DM_E
log_z=np.log10(z)            #np.log10 calculate the log of all the data
points in z array and new array created is stored in log_z

plt.scatter(log_z,log_DM_E,s=4) #Scatter plot between datapoints of
log_DM_E and log_z array
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples (DM_hg_loc=(200,50)).jpg")

poly_deg = 2
p = np.polyfit(log_z,log_DM_E, poly_deg)
print(p)
y_fit = np.poly1d(p) #use np.poly1d to give the polynomial function
corresponding to the coeffs. in p

#Plot the data and the fitted function:
plt.plot(log_z, y_fit(log_z), ls='-', label='polynomial of deg.
{}'.format(poly_deg),color='red')   #to plot the best fit line of degree 2

plt.scatter(log_z,log_DM_E,s=6)    #to plot scatter plot between log_z and
log_DM_E
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples (DM_hg_loc=(200,50)).jpg")
```

Appendix 2.7:

```
pip install pynverse   #install pynverse library which contains
inversefunc which is used to calculate the inverse function
```

```
#importing the necessary libraries
import numpy as np
from scipy import integrate
import math as m
from pynverse import inversefunc
import pandas as pd
import matplotlib.pyplot as plt

#Constants
H_0=67.7   #Hubble Constant
omega_m=0.31
omega_l =0.69   #fraction of the energy of the universe due to the
cosmological constant
omega_b=0.049   #current baryon mass density fraction of the universe
f_IGM= 0.83     #fraction of baryon mass in IGM
K_IGM=933
m_p=1.67*(10**-27)
#P(z)=z*exp(-z)   Probabilty Distribution Function for redshift z
PDF= lambda z : z* m.exp(-z)
CDF= lambda z: 1-m.exp(-z)*(z+1)  # Cumulative Distribution Function

U=np.random.uniform(0,0.80,50)    # Generate 500 data points between 0 and
0.8 uniformly

z=[]
for i in range(len(U)):
  inv_CDF = inversefunc(CDF,U[i])    # Calculate the inverse function
  z=np.append(z,inv_CDF)    # Append the values of z in array z[]

z.sort()  #to sort the z array
z

# Create three empty array DM_IGM,DM_HG_loc and DM_E for storing the
values of DM_IGM,DM_HG_loc and DM_E
DM_IGM=[]
DM_HG_loc=[]
DM_E=[]
function=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+0.69)**0.5
for i in range(0,50):
```

```
  Integration=integrate.quad(function,0,z[i])   #integrate.quad function ,
integrate the given funnction, func from 0 to z
  DM_IGM_f=K_IGM * Integration[0]    #formula to find the value of
DM_IGM,integration[0] stores the result of integration of function , func
  DM_IGM=np.append(DM_IGM,np.random.normal(DM_IGM_f,100))  #introduce a
randomness by taking normal distribution N(DM_IGM,100 pc cm^(-3)) in
DM_IGM
  DM_HG_loc=np.append(DM_HG_loc,np.random.normal(100,20))  #take DM_HG_loc
as normal distribution N(100 pc cm^(-3),20 pc cm^(-3))
  DM_E_formula=DM_IGM[i]+(DM_HG_loc[i]/(1+z[i]))  #formula to calculate
value of DM_E
  DM_E=np.append(DM_E,DM_E_formula)

#print(DM_IGM)
print(DM_HG_loc)
print(DM_E)
log_DM_E=np.log10(DM_E)   # np.log10 calculate the log of all the data
points in DM_E array and new array created is stored in log_DM_E
log_z=np.log10(z)  # np.log10 calculate the log of all the data points in
z array and new array created is stored in log_z
plt.scatter(log_z,log_DM_E)   #Scatter plot between datapoints of log_DM_E
and log_z array
poly_deg = 2
p = np.polyfit(log_z,log_DM_E, poly_deg)
print(p)
y_fit = np.poly1d(p) #use np.poly1d to give the polynomial function
corresponding to the coeffs. in p

#Plot the data and the fitted function:

plt.plot(log_z, y_fit(log_z), ls='-', label='polynomial of deg.
{}'.format(poly_deg),color='red')

plt.scatter(log_z,log_DM_E,s=4)  # to plot scatter plot between log_z and
log_DM_E
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("50 Samples(z_f=3).jpg")
```

```
#Range omega_m : 0-1
#Range K_IGM : 800 -1100
#Range DM_HG_loc : 60-120

#function to find chi_square
def find_chi_square(j,k):
  Sum=0      #Sum stores the values of chi_square for each value of K_IGM
and Omega_matter
  for i in range(50):
    func=lambda z:((7/8)*(1+z))/(j*pow(1+z,3)+(1-j))**0.5
    integration=integrate.quad(func,0,z[i])   #integrate.quad function ,
integrate the given funnction, func from 0 to z
    DM_igm=k * integration[0]    #formula to find the value of
DM_IGM,integration[0] stores the result of integration of function , func
    DM_e= DM_igm+ (77.06/(1+z[i]))   # Value of DM_HG_loc is fixed to
95.76
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum

#create an empty array k_igm to store the values of K_IGM (Range : 800 to
1100)
#create an empty array Omega_matter to store the values of Omega_matter
(Range : 0.001 to 0.500)
#create an empty array chi to store the chi square value corresponding to
each value of K_IGM and Omega_matter
#two parameter fitting using DM_IGM and Omega_matter taking DM_HG_loc
constant
k_igm=[]
Omega_matter=[]
chi=[]


for j in range(1,100):
  o_m= j*0.01
  for k in range(800,1100):
        chi_sq=find_chi_square(o_m,k)  #function call
        chi = np.append(chi,chi_sq)
        k_igm=np.append(k_igm,k)
```

```
        Omega_matter=np.append(Omega_matter,o_m)
chi_min=min(chi)  #find the minimum value in the chi array and store it in chi_min
index= np.where(chi==chi_min)  #find the index where the minimum value is stored in chi array
print(index)
print('k_igm_min:',k_igm[index],'\t','Chi_Square Minimum:',chi_min,'\t','omega_m',Omega_matter[index])       #print the K_IGM and Omega_matter values corresponding to the value of chi square minimum
d1=np.array([k_igm,Omega_matter]).T  #creating a numpy array of K_igm and Omega_matter
df=pd.DataFrame(data=d1, columns=["k_igm","Omega_matter"])  #create a dataframe using numpy array d1
df['chi_square']=chi  #add a column chi_square to the dataframe
display(df)

#to plot the contour plot between K_IGM and Omega_matter

plt.figure(figsize=(10,7))
for i in range(0,len(chi)):
  if (chi_min+2.3-0.4<=chi[i]<=chi_min+2.3+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="green",s=3)
#plot the points (colour:green) which lies between 1-sigma
  if (chi_min+6.17-0.4<=chi[i]<=chi_min+6.17+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="red",s=3)
#plot the points (colour:red) which lies between 2-sigma
  if (chi_min+11.8-0.4<=chi[i]<=chi_min+11.8+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="blue",s=3)
#plot the points (colour:blue) which lies between 3-sigma


# Plot

plt.scatter(k_igm[index],Omega_matter[index],color='black')

plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel('K_IGM',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter Fitting',color='purple',size='20')
```

```
plt.xlim(0,0.6)  #define the range of x axis
plt.ylim(800,1100)  #define the range of y axis

plt.show()
#function to find the chi square
def find_chi_square1(j,l):
  Sum=0    #Sum stores the values of chi_square for each value of
DM_HG_loc and Omega_matter
  for i in range(50):
    func=lambda z:((7/8)*(1+z))/(j*pow(1+z,3)+(1-j))**0.5
    integration=integrate.quad(func,0,z[i])  #integrate.quad function ,
integrate the given funnction, func from 0 to z
    DM_igm=992.75* integration[0]  #formula to find the value of
DM_IGM,integration[0] stores the result of integration of function , func
    DM_e= DM_igm+ (l/(1+z[i]))     #formula of DM_E
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum
#create an empty array DM_HG_loc to store the values of DM_HG_loc (Range :
50 to 160)
#create an empty array Omega_matter_1 to store the values of Omega_matter
(Range : 0.001 to 0.500)
#create an empty array chi_1 to store the chi square value corresponding
to each value of DM_HG_loc and Omega_matter
#two parameter fitting using DM_HG_loc and Omega_matter taking DM_IGM
constant

DM_hg_loc=[]
Omega_matter_1=[]
chi_1=[]


for j in range(1,500):
  o_m_1= j*0.001
  for l in range(50,140):
        chi_sq_1=find_chi_square1(o_m_1,l)   #function call
        chi_1= np.append(chi_1,chi_sq_1)  #append the values chi_square
into chi_1
        DM_hg_loc=np.append(DM_hg_loc,l)  #append the values DM_HG_loc
into DM_HG_loc array
```

```
        Omega_matter_1=np.append(Omega_matter_1,o_m_1)  #append the value
of Omega_matter into Omega_matter_1 array
chi_min_1=min(chi_1)   #find the minimum value in the chi_1 array and
store it in chi_min
index_1= np.where(chi_1==chi_min_1)   #find the index where the minimum
value is stored in chi array
print(index_1)
print('Chi_Square
Minimum:',chi_min_1,'\t',chi_min_1,'\t','omega_m',Omega_matter_1[index_1],
'\tDM_HG_loc',DM_hg_loc[index_1])  #print the DM_HG_loc and Omega_matter
values corresponding to the value of chi square minimum
d2=np.array([DM_hg_loc,Omega_matter_1]).T   #creating a numpy array of
DM_HG_loc and Omega_matter
df2=pd.DataFrame(data=d2, columns=["DM_HG_loc","Omega_matter"])  #create a
dataframe using numpy array d2
df2['chi_square']=chi_1     #add a column chi_square to the dataframe
display(df2)
#to plot the contour plot between DM_HG_loc and Omega_matter
plt.figure(figsize=(10,7))
for i in range(0,len(chi_1)):
  if (chi_min_1+2.3-0.4<=chi_1[i]<=chi_min_1+2.3+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="green",s=3)
#plot the points (colour:green) which lies between 1-sigma
  if (chi_min_1+6.17-0.4<=chi_1[i]<=chi_min_1+6.17+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="red",s=3)
#plot the points (colour:red) which lies between 2-sigma
  if (chi_min_1+11.8-0.4<=chi_1[i]<=chi_min_1+11.8+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="blue",s=3)
#plot the points (colour:blue) which lies between 3-sigma




#Plot
plt.scatter(DM_hg_loc[index_1],Omega_matter_1[index_1],color='black')
plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel('DM_HG_loc',size='18')
```

```
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.xlim(0.20,0.45)   #define the range of x axis
plt.ylim(50,140)  #define the range of y axis
plt.show()
#function to find the chi square
def find_chi_square2(j,l):
  Sum=0  #Sum stores the values of chi_square for each value of DM_HG_loc
and  K_IGM
  for i in range(50):
    func=lambda z:((7/8)*(1+z))/(0.38*pow(1+z,3)+(1-0.38))**0.5
    integration=integrate.quad(func,0,z[i])   #integrate.quad function ,
integrate the given funnction, func from 0 to z
    DM_igm=k* integration[0]  #formula to find the value of DM_IGM,
integration[0] stores the result of integration of function , func
    DM_e= DM_igm+ (l/(1+z[i]))  #formula of DM_E
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum
# Create an empty array DM_HG_loc_1 to store the values of DM_HG_loc
(Range : 60 to 120)
# Create an empty array k_igm_1 to store the values of K_IGM (Range : 800
to 1100)
# Create an empty array chi_2 to store the chi square value corresponding
to each value of DM_HG_loc and k_igm
# Two parameter fitting using DM_HG_loc and k_igm taking Omega_matter
constant

DM_hg_loc_1=[]
k_igm_1=[]
chi_2=[]
for k in range(800,1100):
  for l in range(50,140):
        chi_sq_2=find_chi_square2(k,l)    # Function call
        chi_2= np.append(chi_2,chi_sq_2)  # Append the values chi_square
into chi_2
        k_igm_1=np.append(k_igm_1,k)   # Append the values k_igm into
k_igm_1 array
        DM_hg_loc_1=np.append(DM_hg_loc_1,l)  # Append the value of
Omega_matter into DM_hg_loc_1 array
```

```
chi_min_2=min(chi_2)   # Find the minimum value in the chi_1 array and
store it in chi_min
index_2= np.where(chi_2==chi_min_2)  # Find the index where the minimum
value is stored in chi array
print(index_2)
print('Chi_Square
Minimum:',chi_min_2,'\t','K_IGM',k_igm_1[index_2],'\tDM_HG_loc',DM_hg_loc_
1[index_2])    # Print the DM_HG_loc and k_igm values corresponding to the
value of chi square minimum
d3=np.array([DM_hg_loc_1,k_igm_1]).T  # Creating a numpy array of
DM_hg_loc and K_igm
df3=pd.DataFrame(data=d3, columns=["DM_HG_loc","K_IGM"]) # Create a
dataframe using numpy array d3
df3['chi_square']=chi_2  # Add a column chi_square to the dataframe
display(df3)
# To plot the contour plot between DM_HG_loc and K_IGM
plt.figure(figsize=(10,7))
for i in range(0,len(chi_2)):
  if (chi_min_2+2.3-0.4<=chi_2[i]<=chi_min_2+2.3+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="green",s=3) #
Plot the points (colour:green) which lies between 1-sigma
  if (chi_min_2+6.17-0.4<=chi_2[i]<=chi_min_2+6.17+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="red",s=3) #
Plot the points (colour:red) which lies between 2-sigma
  if (chi_min_2+11.8-0.4<=chi_2[i]<=chi_min_2+11.8+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="blue",s=3) #
Plot the points (colour:blue) which lies between 3-sigma



#Plot
plt.scatter(DM_hg_loc_1[index_2],k_igm_1[index_2],color='black')
plt.xlabel('K_IGM',size='18')
plt.ylabel('DM_HG_loc',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.xlim(800,1100)   # Define the range of x-axis
plt.ylim(50,140)  # Define the range of y-axis
plt.show()
```

Appendix 2.8:

```
pip install pynverse    #install pynverse library which contains
inversefunc which is used to calculate the inverse function

#importing the necessary libraries
import numpy as np
from scipy import integrate
import math as m
from pynverse import inversefunc
import pandas as pd
import matplotlib.pyplot as plt

#Constants
H_0=67.7   #Hubble Constant
omega_m=0.31
omega_l =0.69   #fraction of the energy of the universe due to the
cosmological constant
omega_b=0.049   #current baryon mass density fraction of the universe
f_IGM= 0.83     #fraction of baryon mass in IGM
K_IGM=933
m_p=1.67*(10**-27)

#P(z)=z*exp(-z)    Probabilty Distribution Function for redshift z
PDF= lambda z : z* m.exp(-z)
CDF= lambda z: 1-m.exp(-z)*(z+1)  # Cumulative Distribution Function

U=np.random.uniform(0,0.265,500)   # Generate 500 data points between 0
and 0.8 uniformly
z=[]
for i in range(len(U)):
  inv_CDF = inversefunc(CDF,U[i])  # Calculate the inverse function
  z=np.append(z,inv_CDF)     # Append the values of z in array z[]
z.sort()   #to sort the z array
# Create three empty array DM_IGM,DM_HG_loc and DM_E for storing the
values of DM_IGM,DM_HG_loc and DM_E
DM_IGM=[]
DM_HG_loc=[]
DM_E=[]
function=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+0.69)**0.5
```

```
for i in range(0,500):
  Integration=integrate.quad(function,0,z[i])   #integrate.quad function ,
integrate the given funnction, func from 0 to z
  DM_IGM_f=K_IGM * Integration[0]                 #formula to find the value
of DM_IGM,integration[0] stores the result of integration of function ,
func
  DM_IGM=np.append(DM_IGM,np.random.normal(DM_IGM_f,100))  #introduce a
randomness by taking normal distribution N(DM_IGM,100 pc cm^(-3)) in
DM_IGM
  DM_HG_loc=np.append(DM_HG_loc,np.random.normal(100,20))   #take
DM_HG_loc as normal distribution N(100 pc cm^(-3),20 pc cm^(-3))
  DM_E_formula=DM_IGM[i]+(DM_HG_loc[i]/(1+z[i]))    #formula to calculate
value of DM_E
  DM_E=np.append(DM_E,DM_E_formula)
log_DM_E=np.log10(DM_E)  # np.log10 calculate the log of all the data
points in DM_E array and new array created is stored in log_DM_E
log_z=np.log10(z)    # np.log10 calculate the log of all the data points
in z array and new array created is stored in log_z
'''
plt.figure(figsize=(10,7))
plt.scatter(log_z,log_DM_E,s=6)
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples(z_f=1).jpg")'''
poly_deg = 2
p = np.polyfit(log_z,log_DM_E, poly_deg)
print(p)
y_fit = np.poly1d(p) #use np.poly1d to give the polynomial function
corresponding to the coeffs. in p

#Plot the data and the fitted function:
plt.plot(log_z, y_fit(log_z), ls='-', label='polynomial of deg.
{}'.format(poly_deg),color='red')
plt.ylim(1.6,3.2)
plt.scatter(log_z,log_DM_E,s=6)
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
```

```
plt.show()
plt.savefig("500 Samples(z_f=1).jpg")
#function to find chi_square
def find_chi_square(j,k):
  Sum=0   #Sum stores the values of chi_square for each value of K_IGM and
Omega_matter
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(j*pow(1+z,3)+(1-j))**0.5
    integration=integrate.quad(func,0,z[i])  #integrate.quad function ,
integrate the given funnction, func from 0 to z
    DM_igm=k * integration[0]   #formula to find the value of
DM_IGM,integration[0] stores the result of integration of function , func
    DM_e= DM_igm+ (102.93/(1+z[i]))   # Value of DM_HG_loc is fixed to
95.76
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum
#create an empty array k_igm to store the values of K_IGM (Range : 800 to
1100)
#create an empty array Omega_matter to store the values of Omega_matter
(Range : 0.001 to 0.500)
#create an empty array chi to store the chi square value corresponding to
each value of K_IGM and Omega_matter
#two parameter fitting using DM_IGM and Omega_matter taking DM_HG_loc
constant
k_igm=[]
Omega_matter=[]
chi=[]


for j in range(1,100):
  o_m= j*0.01
  for k in range(800,1100):
    #for l in range(60,120):
        chi_sq=find_chi_square(o_m,k)   #function call
        chi = np.append(chi,chi_sq)
        k_igm=np.append(k_igm,k)

        Omega_matter=np.append(Omega_matter,o_m)
```

```
chi_min=min(chi)                 #find the minimum value in the chi array and
store it in chi_min
index= np.where(chi==chi_min) #find the index where the minimum value is
stored in chi array
print(index)
print('k_igm_min:',k_igm[index],'\t','Chi_Square
Minimum:',chi_min,'\t','omega_m',Omega_matter[index])      #print the K_IGM
and Omega_matter values corresponding to the value of chi square minimum

d1=np.array([k_igm,Omega_matter]).T   #creating a numpy array of K_igm and
Omega_matter
df=pd.DataFrame(data=d1, columns=["k_igm","Omega_matter"])  #create a
dataframe using numpy array d1
df['chi_square']=chi     #add a column chi_square to the dataframe
print(df)
#to plot the contour plot between K_IGM and Omega_matter
plt.figure(figsize=(10,7))
for i in range(0,len(chi)):
  if (chi_min+2.3-0.4<=chi[i]<=chi_min+2.3+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="green",s=3)
#plot the points (colour:green) which lies between 1-sigma
  if (chi_min+6.17-0.4<=chi[i]<=chi_min+6.17+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="red",s=3)
#plot the points (colour:red) which lies between 2-sigma
  if (chi_min+11.8-0.4<=chi[i]<=chi_min+11.8+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="blue",s=3)
#plot the points (colour:blue) which lies between 3-sigma



# Plot
plt.scatter(k_igm[index],Omega_matter[index],color='black')

plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel('K_IGM',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.xlim(0.00,0.50)  #define the range of x axis
plt.ylim(800,1100)   #define the range of y axis
plt.show()
```

```
#function to find the chi square
def find_chi_square1(j,l):
  Sum=0     #Sum stores the values of chi_square for each value of
DM_HG_loc and Omega_matter `
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(j*pow(1+z,3)+(1-j))**0.5
    integration=integrate.quad(func,0,z[i])   #integrate.quad function ,
integrate the given funnction, func from 0 to z
    #taking the value of k_igm as 937.05
    DM_igm=931.04* integration[0]      #formula to find the value of
DM_IGM,integration[0] stores the result of integration of function , func
    DM_e= DM_igm+ (l/(1+z[i]))     #formula of DM_E
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum
#create an empty array DM_HG_loc to store the values of DM_HG_loc (Range :
50 to 160)
#create an empty array Omega_matter_1 to store the values of Omega_matter
(Range : 0.001 to 0.500)
#create an empty array chi_1 to store the chi square value corresponding
to each value of DM_HG_loc and Omega_matter
#two parameter fitting using DM_HG_loc and Omega_matter taking DM_IGM
constant

DM_hg_loc=[]
Omega_matter_1=[]
chi_1=[]


for j in range(1,500):
  o_m_1= j*0.001
  #for k in range(800,1100):
  for l in range(60,120):
        chi_sq_1=find_chi_square1(o_m_1,l)    #function call
        chi_1= np.append(chi_1,chi_sq_1)   #append the values chi_square
into chi_1
        DM_hg_loc=np.append(DM_hg_loc,l)     #append the values DM_HG_loc
into DM_HG_loc array
        Omega_matter_1=np.append(Omega_matter_1,o_m_1)    #append the
value of Omega_matter into Omega_matter_1 array
```

```
chi_min_1=min(chi_1)       #find the minimum value in the chi_1 array and
store it in chi_min
index_1= np.where(chi_1==chi_min_1)    #find the index where the minimum
value is stored in chi array
print(index_1)
print('Chi_Square
Minimum:',chi_min_1,'\t','omega_m',Omega_matter_1[index_1],'\tDM_HG_loc',D
M_hg_loc[index_1])     #print the DM_HG_loc and Omega_matter values
corresponding to the value of chi square minimum
d2=np.array([DM_hg_loc,Omega_matter_1]).T       #creating a numpy array of
DM_HG_loc and Omega_matter
df2=pd.DataFrame(data=d2, columns=["DM_HG_loc","Omega_matter"])
#create a dataframe using numpy array d2
df2['chi_square']=chi_1       #add a column chi_square to the dataframe
display(df2)
#to plot the contour plot between DM_HG_loc and Omega_matter
plt.figure(figsize=(10,7))
for i in range(0,len(chi_1)):
  if (chi_min_1+2.3-0.4<=chi_1[i]<=chi_min_1+2.3+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="green",s=3)
#plot the points (colour:green) which lies between 1-sigma
  if (chi_min_1+6.17-0.4<=chi_1[i]<=chi_min_1+6.17+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="red",s=3)
#plot the points (colour:red) which lies between 2-sigma
  if (chi_min_1+11.8-0.4<=chi_1[i]<=chi_min_1+11.8+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="blue",s=3)
#plot the points (colour:blue) which lies between 3-sigma



#Plot
plt.scatter(DM_hg_loc[index_1],Omega_matter_1[index_1],color='black')
plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel('DM_HG_loc',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.xlim(0.2,0.4)   #define the range of x axis
```

```
plt.ylim(60,120)    #define the range of y axis
plt.show()
#function to find the chi square
def find_chi_square2(k,l):
  Sum=0         #Sum stores the values of chi_square for each value of 
DM_HG_loc and  K_IGM
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+(1-0.31))**0.5
    integration=integrate.quad(func,0,z[i])          #integrate.quad 
function , integrate the given funnction, func from 0 to z
    DM_igm=k* integration[0]                #formula to find the value of 
DM_IGM, integration[0] stores the result of integration of function , func
    DM_e= DM_igm+ (l/(1+z[i]))                    #formula of DM_E
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum
# Create an empty array DM_HG_loc_1 to store the values of DM_HG_loc 
(Range : 60 to 120)
# Create an empty array k_igm_1 to store the values of K_IGM (Range : 800 
to 1100)
# Create an empty array chi_2 to store the chi square value corresponding 
to each value of DM_HG_loc and k_igm
# Two parameter fitting using DM_HG_loc and k_igm taking Omega_matter 
constant


DM_hg_loc_1=[]
k_igm_1=[]
chi_2=[]


for k in range(800,1100):
  for l in range(60,120):
        chi_sq_2=find_chi_square2(k,l)   # Function call
        chi_2= np.append(chi_2,chi_sq_2)    # Append the values chi_square 
into chi_2
        k_igm_1=np.append(k_igm_1,k)       # Append the values k_igm into 
k_igm_1 array
        DM_hg_loc_1=np.append(DM_hg_loc_1,l)        # Append the value of 
Omega_matter into DM_hg_loc_1 array
```

```
chi_min_2=min(chi_2)              # Find the minimum value in the chi_1
array and store it in chi_min
index_2= np.where(chi_2==chi_min_2)           # Find the index where the
minimum value is stored in chi array
print(index_2)
print('Chi_Square
Minimum:',chi_min_2,'\t','K_IGM',k_igm_1[index_2],'\tDM_HG_loc',DM_hg_loc_
1[index_2])      # Print the DM_HG_loc and k_igm values corresponding to
the value of chi square minimum
d3=np.array([DM_hg_loc_1,k_igm_1]).T    # Creating a numpy array of
DM_hg_loc and K_igm
df3=pd.DataFrame(data=d3, columns=["DM_HG_loc","K_IGM"])     # Create a
dataframe using numpy array d3
df3['chi_square']=chi_2     # Add a column chi_square to the dataframe
display(df3)
# To plot the contour plot between DM_HG_loc and K_IGM
plt.figure(figsize=(10,7))
for i in range(0,len(chi_2)):
  if (chi_min_2+2.3-0.4<=chi_2[i]<=chi_min_2+2.3+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="green",s=3)   #
Plot the points (colour:green) which lies between 1-sigma
  if (chi_min_2+6.17-0.4<=chi_2[i]<=chi_min_2+6.17+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="red",s=3)  #
Plot the points (colour:red) which lies between 2-sigma
  if (chi_min_2+11.8-0.4<=chi_2[i]<=chi_min_2+11.8+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="blue",s=3)   #
Plot the points (colour:blue) which lies between 3-sigma



#Plot
plt.scatter(DM_hg_loc_1[index_2],k_igm_1[index_2],color='black')
plt.xlim(800,1100)    # Define the range of x-axis
plt.ylim(60,120)     # Define the range of y-axis
plt.xlabel('K_IGM',size='18')
plt.ylabel('DM_HG_loc',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.show()
```

Appendix 2.9:

```
pip install pynverse    #install pynverse library which contains
inversefunc which is used to calculate the inverse function

#importing the necessary libraries
import numpy as np
from scipy import integrate
import math as m
from pynverse import inversefunc
import pandas as pd
import matplotlib.pyplot as plt

# Constants
H_0=67.7   # Hubble Constant
omega_m=0.31
omega_l =0.69   # Fraction of the energy of the universe due to the
cosmological constant
omega_b=0.049   # Current baryon mass density fraction of the universe
f_IGM= 0.83     # Fraction of baryon mass in IGM
K_IGM=933
m_p=1.67*(10**-27)
#P(z)=z*exp(-z)     Probabilty Distribution Function for redshift z
PDF= lambda z : z* m.exp(-z)
CDF= lambda z: 1-m.exp(-z)*(z+1)    # Cumulative Distribution Function

U=np.random.uniform(0,0.8,500)   # Generate 500 data points between 0 and
0.8 uniformly

z=[]
for i in range(len(U)):
  inv_CDF = inversefunc(CDF,U[i])    # Calculate the inverse function
  z=np.append(z,inv_CDF)             # Append the values of z in array z[]

z.sort()    #to sort the z array

# Create three empty array DM_IGM,DM_HG_loc and DM_E for storing the
values of DM_IGM,DM_HG_loc and DM_E
DM_IGM=[]
DM_HG_loc=[]
```

```
DM_E=[]
function=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+0.69)**0.5
for i in range(0,500):
  Integration=integrate.quad(function,0,z[i])     #integrate.quad function
, integrate the given funnction, func from 0 to z
  DM_IGM_f=K_IGM * Integration[0]                  #formula to find the
value of DM_IGM,integration[0] stores the result of integration of
function , func
  DM_IGM=np.append(DM_IGM,np.random.normal(DM_IGM_f,100))     #introduce a
randomness by taking normal distribution N(DM_IGM,100 pc cm^(-3)) in
DM_IGM
  DM_HG_loc=np.append(DM_HG_loc,np.random.normal(100,20))     #take
DM_HG_loc as normal distribution N(100 pc cm^(-3),20 pc cm^(-3))
  DM_E_formula=DM_IGM[i]+(DM_HG_loc[i]/(1+z[i]))              #formula to
calculate value of DM_E
  DM_E=np.append(DM_E,DM_E_formula)



log_DM_E=np.log10(DM_E)    # np.log10 calculate the log of all the data
points in DM_E array and new array created is stored in log_DM_E
log_z=np.log10(z)          # np.log10 calculate the log of all the data
points in z array and new array created is stored in log_z

plt.scatter(log_z,log_DM_E,s=4)  #Scatter plot between datapoints of
log_DM_E and log_z array
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples(z_f=3).jpg")



poly_deg = 2
p = np.polyfit(log_z,log_DM_E, poly_deg)
print(p)
y_fit = np.poly1d(p) #use np.poly1d to give the polynomial function
corresponding to the coeffs. in p

#Plot the data and the fitted function:
```

```
plt.plot(log_z, y_fit(log_z), ls='-', label='polynomial of deg.
{}'.format(poly_deg),color='red')   #to plot the best fit line of degree 2

plt.scatter(log_z,log_DM_E,s=4) # to plot scatter plot between log_z and
log_DM_E
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples(z_f=3).jpg")


#function to find chi_square
def find_chi_square(j,k):
  Sum=0                 #Sum stores the values of chi_square for each
value of K_IGM and Omega_matter
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(j*pow(1+z,3)+(1-j))**0.5
    integration=integrate.quad(func,0,z[i])          #integrate.quad
function , integrate the given funnction, func from 0 to z
    DM_igm=k * integration[0]                        #formula to find the
value of DM_IGM,integration[0] stores the result of integration of
function , func
    DM_e= DM_igm+ (95.76/(1+z[i]))                   # Value of DM_HG_loc
is fixed to 95.76
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum

#create an empty array k_igm to store the values of K_IGM (Range : 800 to
1100)
#create an empty array Omega_matter to store the values of Omega_matter
(Range : 0.001 to 0.500)
#create an empty array chi to store the chi square value corresponding to
each value of K_IGM and Omega_matter
#two parameter fitting using DM_IGM and Omega_matter taking DM_HG_loc
constant
k_igm=[]
Omega_matter=[]
```

```
chi=[]


for j in range(1,500):
  o_m= j*0.001
  for k in range(800,1100):
        chi_sq=find_chi_square(o_m,k)        #function call
        chi = np.append(chi,chi_sq)
        k_igm=np.append(k_igm,k)
        Omega_matter=np.append(Omega_matter,o_m)
chi_min=min(chi)                              #find the minimum value in the
chi array and store it in chi_min
index= np.where(chi==chi_min)                 #find the index where the
minimum value is stored in chi array
print(index)
print('k_igm_min:',k_igm[index],'\t','Chi_Square
Minimum:',chi_min,'\t','omega_m',Omega_matter[index])   #print the K_IGM
and Omega_matter values corresponding to the value of chi square minimum


d1=np.array([k_igm,Omega_matter]).T      #creating a numpy array of K_igm
and Omega_matter
df=pd.DataFrame(data=d1, columns=["k_igm","Omega_matter"])      #create a
dataframe using numpy array d1
df['chi_square']=chi        #add a column chi_square to the dataframe
display(df)

#to plot the contour plot between K_IGM and Omega_matter

plt.figure(figsize=(10,7))
for i in range(0,len(chi)):
  if (chi_min+2.3-0.4<=chi[i]<=chi_min+2.3+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="green",s=3)
#plot the points (colour:green) which lies between 1-sigma
  if (chi_min+6.17-0.4<=chi[i]<=chi_min+6.17+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="red",s=3)
#plot the points (colour:red) which lies between 2-sigma
  if (chi_min+11.8-0.4<=chi[i]<=chi_min+11.8+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="blue",s=3)
#plot the points (colour:blue) which lies between 3-sigma
```

```
#Plot
plt.scatter(k_igm[index],Omega_matter[index],color='black')

plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel('K_IGM',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='18')
plt.xlim(0.25,0.40)     #define the range of x axis
plt.xticks([0.25,0.3,0.35,0.4])
plt.ylim(850,1050)     #define the range of y axis
plt.show()


#function to find the chi square
def find_chi_square1(j,l):
  Sum=0                `     #Sum stores the values of chi_square for each
value of DM_HG_loc and Omega_matter `
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(j*pow(1+z,3)+(1-j))**0.5
    integration=integrate.quad(func,0,z[i])       #integrate.quad function
, integrate the given funnction, func from 0 to z
    #taking the value of k_igm as 937.05
    DM_igm=937.05* integration[0]                  #formula to find the
value of DM_IGM,integration[0] stores the result of integration of
function , func
    DM_e= DM_igm+ (l/(1+z[i]))                     #formula of DM_E
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum

#create an empty array DM_HG_loc to store the values of DM_HG_loc (Range :
50 to 160)
#create an empty array Omega_matter_1 to store the values of Omega_matter
(Range : 0.001 to 0.500)
#create an empty array chi_1 to store the chi square value corresponding
to each value of DM_HG_loc and Omega_matter
```

```
#two parameter fitting using DM_HG_loc and Omega_matter taking DM_IGM
constant

DM_hg_loc=[]
Omega_matter_1=[]
chi_1=[]


for j in range(1,500):
  o_m_1= j*0.001
  for l in range(50,160):
        chi_sq_1=find_chi_square1(o_m_1,l)    #function call
        chi_1= np.append(chi_1,chi_sq_1)      #append the values
chi_square into chi_1
        DM_hg_loc=np.append(DM_hg_loc,l)      #append the values DM_HG_loc
into DM_HG_loc array
        Omega_matter_1=np.append(Omega_matter_1,o_m_1)   #append the value
of Omega_matter into Omega_matter_1 array
chi_min_1=min(chi_1)                             #find the minimum value in
the chi_1 array and store it in chi_min_1
index_1= np.where(chi_1==chi_min_1)              #find the index where the
minimum value is stored in chi array
print(index_1)
print('Chi_Square
Minimum:',chi_min_1,'\t','omega_m',Omega_matter_1[index_1],'\tDM_HG_loc',D
M_hg_loc[index_1])  #print the DM_HG_loc and Omega_matter values
corresponding to the value of chi square minimum


d2=np.array([DM_hg_loc,Omega_matter_1]).T    #creating a numpy array of
DM_HG_loc and Omega_matter
df2=pd.DataFrame(data=d2, columns=["DM_HG_loc","Omega_matter"])    #create
a dataframe using numpy array d2
df2['chi_square']=chi_1       #add a column chi_square to the dataframe
display(df2)


#to plot the contour plot between DM_HG_loc and Omega_matter
plt.figure(figsize=(10,7))
for i in range(0,len(chi_1)):
```

```
  if (chi_min_1+2.3-0.4<=chi_1[i]<=chi_min_1+2.3+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="green",s=3)
#plot the points (colour:green) which lies between 1-sigma
  if (chi_min_1+6.17-0.4<=chi_1[i]<=chi_min_1+6.17+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="red",s=3)
#plot the points (colour:red) which lies between 2-sigma
  if (chi_min_1+11.8-0.4<=chi_1[i]<=chi_min_1+11.8+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="blue",s=3)
#plot the points (colour:blue) which lies between 3-sigma


#Plot
plt.scatter(DM_hg_loc[index_1],Omega_matter_1[index_1],color='black')
plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel('DM_HG_loc',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.xlim(0.29,0.34)    #define the range of x axis
plt.ylim(60,130)       #define the range of y axis
plt.show()


#function to find the chi square
def find_chi_square2(j,l):
  Sum=0                              #Sum stores the values of chi_square for
each value of DM_HG_loc and  K_IGM
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+(1-0.31))**0.5
    integration=integrate.quad(func,0,z[i])
#integrate.quad function , integrate the given funnction, func from 0 to z
    DM_igm=k* integration[0]
#formula to find the value of DM_IGM, integration[0] stores the result of
integration of function , func
    DM_e= DM_igm+ (l/(1+z[i]))
#formula of DM_E
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
```

```
  return Sum


# Create an empty array DM_HG_loc_1 to store the values of DM_HG_loc
(Range : 60 to 120)
# Create an empty array k_igm_1 to store the values of K_IGM (Range : 800
to 1100)
# Create an empty array chi_2 to store the chi square value corresponding
to each value of DM_HG_loc and k_igm
# Two parameter fitting using DM_HG_loc and k_igm taking Omega_matter
constant

DM_hg_loc_1=[]
k_igm_1=[]
chi_2=[]



for k in range(800,1100):
  for l in range(60,120):
        chi_sq_2=find_chi_square2(k,l)              # Function call
        chi_2= np.append(chi_2,chi_sq_2)            # Append the values
chi_square into chi_2
        k_igm_1=np.append(k_igm_1,k)                # Append the values
k_igm into k_igm_1 array
        DM_hg_loc_1=np.append(DM_hg_loc_1,l)        # Append the value of
Omega_matter into DM_hg_loc_1 array

chi_min_2=min(chi_2)                          # Find the minimum value in
the chi_2 array and store it in chi_min_2
index_2= np.where(chi_2==chi_min_2)           # Find the index where the
minimum value is stored in chi array
print(index_2)
print('Chi_Square
Minimum:',chi_min_2,'\t','K_IGM',k_igm_1[index_2],'\tDM_HG_loc',DM_hg_loc_
1[index_2])          # Print the DM_HG_loc and k_igm values corresponding
to the value of chi square minimum

d3=np.array([DM_hg_loc_1,k_igm_1]).T # Creating a numpy array of DM_hg_loc
and K_igm
```

```
df3=pd.DataFrame(data=d3, columns=["DM_HG_loc","K_IGM"])       # Create a
dataframe using numpy array d3
df3['chi_square']=chi_2    # Add a column chi_square to the dataframe
display(df3)


plt.figure(figsize=(10,7))

# To plot the contour plot between DM_HG_loc and K_IGM
for i in range(0,len(chi_2)):
  if (chi_min_2+2.3-0.4<=chi_2[i]<=chi_min_2+2.3+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="green",s=3)
# Plot the points (colour:green) which lies between 1-sigma
  if (chi_min_2+6.17-0.4<=chi_2[i]<=chi_min_2+6.17+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="red",s=3)
# Plot the points (colour:red) which lies between 2-sigma
  if (chi_min_2+11.8-0.4<=chi_2[i]<=chi_min_2+11.8+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="blue",s=3)
# Plot the points (colour:blue) which lies between 3-sigma


# Plot
plt.scatter(DM_hg_loc_1[index_2],k_igm_1[index_2],color='black')
plt.xlabel('K_IGM',size='18')
plt.ylabel('DM_HG_loc',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.ylim(60,120)      # Define the range of x-axis
plt.xlim(900,1000)    # Define the range of y-axis
plt.show()
```

<u>Appendix 3.0:</u>

```
pip install pynverse     #install pynverse library which contains
inversefunc which is used to calculate the inverse function

#importing the necessary libraries
import numpy as np
from scipy import integrate
import math as m
```

```
from pynverse import inversefunc
import pandas as pd
import matplotlib.pyplot as plt

#Constants
H_0=67.7   #Hubble Constant
omega_m=0.31
omega_l =0.69   #fraction of the energy of the universe due to the
cosmological constant
omega_b=0.049   #current baryon mass density fraction of the universe
f_IGM= 0.83     #fraction of baryon mass in IGM
K_IGM=933
m_p=1.67*(10**-27)

#P(z)=z*exp(-z)          Probabilty Distribution Function for redshift z
PDF= lambda z : z* m.exp(-z)
CDF= lambda z: 1-m.exp(-z)*(z+1)       # Cumulative Distribution Function

U=np.random.uniform(0,0.594,500)       # Generate 500 data points between
0 and 0.8 uniformly
z=[]
for i in range(len(U)):
  inv_CDF = inversefunc(CDF,U[i])      # Calculate the inverse function
  z=np.append(z,inv_CDF)               # Append the values of z in array
z[]
z.sort()                           #to sort the z array

#Create three empty array DM_IGM,DM_HG_loc and DM_E for storing the values
of DM_IGM,DM_HG_loc and DM_E
DM_IGM=[]
DM_HG_loc=[]
DM_E=[]
function=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+0.69)**0.5
for i in range(0,500):
  Integration=integrate.quad(function,0,z[i])       #integrate.quad
function , integrate the given funnction, func from 0 to z
  DM_IGM_f=K_IGM * Integration[0]                   #formula to find the
value of DM_IGM,integration[0] stores the result of integration of
function , func
```

```
  DM_IGM=np.append(DM_IGM,np.random.normal(DM_IGM_f,100))    #introduce a
randomness by taking normal distribution N(DM_IGM,100 pc cm^(-3)) in
DM_IGM
  DM_HG_loc=np.append(DM_HG_loc,np.random.normal(100,20))    #take
DM_HG_loc as normal distribution N(100 pc cm^(-3),20 pc cm^(-3))
  DM_E_formula=DM_IGM[i]+(DM_HG_loc[i]/(1+z[i]))            #formula to
calculate value of DM_E
  DM_E=np.append(DM_E,DM_E_formula)

log_DM_E=np.log10(DM_E)      # np.log10 calculate the log of all the data
points in DM_E array and new array created is stored in log_DM_E
log_z=np.log10(z)            # np.log10 calculate the log of all the data
points in z array and new array created is stored in log_z

plt.figure(figsize=(10,7))
plt.scatter(log_z,log_DM_E,s=6)   #Scatter plot between datapoints of
log_DM_E and log_z array
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples(z_f=2).jpg")


poly_deg = 2
p = np.polyfit(log_z,log_DM_E, poly_deg)
print(p)
y_fit = np.poly1d(p) #use np.poly1d to give the polynomial function
corresponding to the coeffs. in p

#Plot the data and the fitted function:
plt.plot(log_z, y_fit(log_z), ls='-', label='polynomial of deg.
{}'.format(poly_deg),color='red')

plt.scatter(log_z,log_DM_E,s=6)       # to plot scatter plot between log_z
and log_DM_E
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
```

```
plt.savefig("500 Samples(z_f=2).jpg")


#Range omega_m : 0-1
#Range K_IGM : 800 -1100
#Range DM_HG_loc : 60-120
#function to find chi_square

def find_chi_square(j,k):
  Sum=0                              #Sum stores the values of chi_square
for each value of K_IGM and Omega_matter
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(j*pow(1+z,3)+(1-j))**0.5
    integration=integrate.quad(func,0,z[i])           #integrate.quad
function , integrate the given funnction, func from 0 to z
    DM_igm=k * integration[0]                         #formula to find the
value of DM_IGM,integration[0] stores the result of integration of
function , func
    DM_e= DM_igm+ (93.30/(1+z[i]))                    # Value of DM_HG_loc
is fixed to 93.30
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum


#create an empty array k_igm to store the values of K_IGM (Range : 800 to
1100)
#create an empty array Omega_matter to store the values of Omega_matter
(Range : 0.01 to 1.00)
#create an empty array chi to store the chi square value corresponding to
each value of K_IGM and Omega_matter
#two parameter fitting using DM_IGM and Omega_matter taking DM_HG_loc
constant
k_igm=[]
Omega_matter=[]
chi=[]


for j in range(1,100):
  o_m= j*0.01
```

```
  for k in range(800,1100):

        chi_sq=find_chi_square(o_m,k)          #function call
        chi = np.append(chi,chi_sq)
        k_igm=np.append(k_igm,k)
        Omega_matter=np.append(Omega_matter,o_m)
chi_min=min(chi)                                          #find the minimum value
in the chi array and store it in chi_min
index= np.where(chi==chi_min)                             #find the index where
the minimum value is stored in chi array
print(index)
print('k_igm_min:',k_igm[index],'\t','Chi_Square
Minimum:',chi_min,'\tomega_m',Omega_matter[index])  #print the K_IGM and
Omega_matter values corresponding to the value of chi square minimum


d1=np.array([k_igm,Omega_matter]).T                      #creating a numpy array
of K_igm and Omega_matter
df=pd.DataFrame(data=d1, columns=["k_igm","Omega_matter"])        #create
a dataframe using numpy array d1
df['chi_square']=chi          #add a column chi_square to the dataframe
display(df)


#to plot the contour plot between K_IGM and Omega_matter

plt.figure(figsize=(10,7))
for i in range(0,len(chi)):
  if (chi_min+2.3-0.4<=chi[i]<=chi_min+2.3+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="green",s=3)
#plot the points (colour:green) which lies between 1-sigma
  if (chi_min+6.17-0.4<=chi[i]<=chi_min+6.17+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="red",s=3)
#plot the points (colour:red) which lies between 2-sigma
  if (chi_min+11.8-0.4<=chi[i]<=chi_min+11.8+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="blue",s=3)
#plot the points (colour:blue) which lies between 3-sigma


# Plot
```

```
plt.scatter(k_igm[index],Omega_matter[index],color='black')


plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel('K_IGM',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.xlim(0.25,0.50)          #define the range of x axis
plt.ylim(800,1100)           #define the range of y axis
plt.show()

#function to find the chi square
def find_chi_square1(j,l):
  Sum=0                                  #Sum stores the values of
chi_square for each value of DM_HG_loc and Omega_matter
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(j*pow(1+z,3)+(1-j))**0.5
    integration=integrate.quad(func,0,z[i])        #integrate.quad
function , integrate the given funnction, func from 0 to z
    #taking the value of k_igm as 932.47
    DM_igm=932.47* integration[0]                  #formula to find the
value of DM_IGM,integration[0] stores the result of integration of
function , func
    DM_e= DM_igm+ (l/(1+z[i]))                     #formula of DM_E
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum

#create an empty array DM_HG_loc to store the values of DM_HG_loc (Range :
60 to 120)
#create an empty array Omega_matter_1 to store the values of Omega_matter
(Range : 0.001 to 0.500)
#create an empty array chi_1 to store the chi square value corresponding
to each value of DM_HG_loc and Omega_matter
#two parameter fitting using DM_HG_loc and Omega_matter taking DM_IGM
constant

DM_hg_loc=[]
Omega_matter_1=[]
chi_1=[]
```

```
for j in range(1,500):
  o_m_1= j*0.001
  for l in range(60,120):
        chi_sq_1=find_chi_square1(o_m_1,l)      #function call
        chi_1= np.append(chi_1,chi_sq_1)        #append the values
chi_square into chi_1
        DM_hg_loc=np.append(DM_hg_loc,l)        #append the values
DM_HG_loc into DM_HG_loc array
        Omega_matter_1=np.append(Omega_matter_1,o_m_1)    #append the
value of Omega_matter into Omega_matter_1 array
chi_min_1=min(chi_1)                           #find the minimum value in
the chi_1 array and store it in chi_min_1
index_1= np.where(chi_1==chi_min_1)            #find the index where the
minimum value is stored in chi array
print(index_1)
print('Chi_Square
Minimum:',chi_min_1,'\t','omega_m',Omega_matter_1[index_1],'\tDM_HG_loc',D
M_hg_loc[index_1])    #print the DM_HG_loc and Omega_matter values
corresponding to the value of chi square minimum

d2=np.array([DM_hg_loc,Omega_matter_1]).T              #creating a numpy
array of DM_HG_loc and Omega_matter
df2=pd.DataFrame(data=d2, columns=["DM_HG_loc","Omega_matter"])
#create a dataframe using numpy array d2
df2['chi_square']=chi_1           #add a column chi_square to the
dataframe
display(df2)
#to plot the contour plot between DM_HG_loc and Omega_matter
plt.figure(figsize=(10,7))
for i in range(0,len(chi_1)):
  if (chi_min_1+2.3-0.4<=chi_1[i]<=chi_min_1+2.3+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="green",s=3)
#plot the points (colour:green) which lies between 1-sigma
  if (chi_min_1+6.17-0.4<=chi_1[i]<=chi_min_1+6.17+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="red",s=3)
#plot the points (colour:red) which lies between 2-sigma
  if (chi_min_1+11.8-0.4<=chi_1[i]<=chi_min_1+11.8+0.4):
```

```
plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="blue",s=3)
#plot the points (colour:blue) which lies between 3-sigma


#Plot
plt.scatter(DM_hg_loc[index_1],Omega_matter_1[index_1],color='black')
plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel('DM_HG_loc',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.xlim(0.2,0.4)       #define the range of x axis
plt.ylim(60,120)       #define the range of y axis
plt.show()

#function to find the chi square
#take Omega_matter to be equal to 0.31
def find_chi_square2(k,l):
  Sum=0                            #Sum stores the values of chi_square
for each value of DM_HG_loc and  K_IGM
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+(1-0.31))**0.5
    integration=integrate.quad(func,0,z[i])          #integrate.quad
function , integrate the given funnction, func from 0 to z
    DM_igm=k* integration[0]                         #formula to find the
value of DM_IGM, integration[0] stores the result of integration of
function , func
    DM_e= DM_igm+ (l/(1+z[i]))                       #formula of DM_E
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum

# Create an empty array DM_HG_loc_1 to store the values of DM_HG_loc
(Range : 60 to 120)
# Create an empty array k_igm_1 to store the values of K_IGM (Range : 800
to 1100)
# Create an empty array chi_2 to store the chi square value corresponding
to each value of DM_HG_loc and k_igm
# Two parameter fitting using DM_HG_loc and k_igm taking Omega_matter
constant
```

```
DM_hg_loc_1=[]
k_igm_1=[]
chi_2=[]


for k in range(800,1100):
  for l in range(60,120):
        chi_sq_2=find_chi_square2(k,l)        # Function call
        chi_2= np.append(chi_2,chi_sq_2)      # Append the values
chi_square into chi_2
        k_igm_1=np.append(k_igm_1,k)          # Append the values k_igm
into k_igm_1 array
        DM_hg_loc_1=np.append(DM_hg_loc_1,l)  # Append the value of
Omega_matter into DM_hg_loc_1 array

chi_min_2=min(chi_2)                           # Find the minimum value in
the chi_2 array and store it in chi_min_2
index_2= np.where(chi_2==chi_min_2)            # Find the index where the
minimum value is stored in chi_2 array
print(index_2)
print('Chi_Square
Minimum:',chi_min_2,'\t','K_IGM',k_igm_1[index_2],'\tDM_HG_loc',DM_hg_loc_
1[index_2])  # Print the DM_HG_loc and k_igm values corresponding to the
value of chi square minimum

d3=np.array([DM_hg_loc_1,k_igm_1]).T    # Creating a numpy array of
DM_hg_loc and K_igm
df3=pd.DataFrame(data=d3, columns=["DM_HG_loc","K_IGM"])   # Create a
dataframe using numpy array d3
df3['chi_square']=chi_2     # Add a column chi_square to the dataframe
display(df3)

plt.figure(figsize=(10,7))
#To plot the contour plot between DM_HG_loc and K_IGM
for i in range(0,len(chi_2)):
  if (chi_min_2+2.3-0.4<=chi_2[i]<=chi_min_2+2.3+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="green",s=3)
# Plot the points (colour:green) which lies between 1-sigma
  if (chi_min_2+6.17-0.4<=chi_2[i]<=chi_min_2+6.17+0.4):
```

```
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="red",s=3)
# Plot the points (colour:red) which lies between 2-sigma
  if (chi_min_2+11.8-0.4<=chi_2[i]<=chi_min_2+11.8+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="blue",s=3)
# Plot the points (colour:blue) which lies between 3-sigma
```

```
#Plot
plt.scatter(DM_hg_loc_1[index_2],k_igm_1[index_2],color='black')
plt.xlabel('K_IGM',size='18')
plt.ylabel('DM_HG_loc',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.xlim(800,1100)      # Define the range of x-axis
plt.ylim(60,120)            # Define the range of y-axis
plt.show()
```

Appendix 3.1:

```
pip install pynverse    #install pynverse library which contains
inversefunc which is used to calculate the inverse function

#importing the necessary libraries
import numpy as np
from scipy import integrate
import math as m
from pynverse import inversefunc
import pandas as pd
import matplotlib.pyplot as plt

#Constants
H_0=67.7   #Hubble Constant
omega_m=0.31
omega_l =0.69   #fraction of the energy of the universe due to the
cosmological constant
omega_b=0.049   #current baryon mass density fraction of the universe
f_IGM= 0.83     #fraction of baryon mass in IGM
K_IGM=933
m_p=1.67*(10**-27)
```

```
#P(z)=z*exp(-z)      Probabilty Distribution Function for redshift z
PDF= lambda z : z* m.exp(-z)
CDF= lambda z: 1-m.exp(-z)*(z+1)  #Cumulative Distribution Function

U=np.random.uniform(0,0.594,500)  #Generate 500 data points between 0 and
0.8 uniformly

z=[]
for i in range(len(U)):
  inv_CDF = inversefunc(CDF,U[i])    # Calculate the inverse function
  z=np.append(z,inv_CDF)             # Append the values of z in array z[]

z.sort()    #to sort the z array

#Create three empty array DM_IGM,DM_HG_loc and DM_E for storing the values
of DM_IGM,DM_HG_loc and DM_E
DM_IGM=[]
DM_HG_loc=[]
DM_E=[]
function=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+0.69)**0.5
for i in range(0,500):
  Integration=integrate.quad(function,0,z[i])
#integrate.quad function , integrate the given funnction, func from 0 to z
  DM_IGM_f=K_IGM * Integration[0]
#formula to find the value of DM_IGM,integration[0] stores the result of
integration of function , func
  DM_IGM=np.append(DM_IGM,np.random.normal(DM_IGM_f,100))

#introduce a randomness by taking normal distribution N(DM_IGM,100 pc
cm^(-3)) in DM_IGM
  DM_HG_loc=np.append(DM_HG_loc,np.random.normal(100,50))
#take DM_HG_loc as normal distribution N(100 pc cm^(-3),20 pc cm^(-3))
  DM_E_formula=DM_IGM[i]+(DM_HG_loc[i]/(1+z[i]))
#formula to calculate value of DM_E
  DM_E=np.append(DM_E,DM_E_formula)

log_DM_E=np.log10(DM_E)
# np.log10 calculate the log of all the data points in DM_E array and new
array created is stored in log_DM_E
log_z=np.log10(z)
```

```
# np.log10 calculate the log of all the data points in z array and new
array created is stored in log_z

plt.scatter(log_z,log_DM_E,s=4)
#Scatter plot between datapoints of log_DM_E and log_z array
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples (DM_hg_loc=(100,50)).jpg")

poly_deg = 2
p = np.polyfit(log_z,log_DM_E, poly_deg)
print(p)
y_fit = np.poly1d(p)
#use np.poly1d to give the polynomial function corresponding to the
coeffs. in p

#Plot the data and the fitted function:
plt.plot(log_z, y_fit(log_z), ls='-', label='polynomial of deg.
{}'.format(poly_deg),color='red')  #to plot the best fit line of degree 2

plt.scatter(log_z,log_DM_E,s=4)
# to plot scatter plot between log_z and log_DM_E
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples (DM_hg_loc=(100,50)).jpg")

#function to find chi_square
def find_chi_square(j,k):
  Sum=0
#Sum stores the values of chi_square for each value of K_IGM and
Omega_matter
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(j*pow(1+z,3)+(1-j))**0.5
    integration=integrate.quad(func,0,z[i])
#integrate.quad function , integrate the given funnction, func from 0 to z
    DM_igm=k * integration[0]
```

```
#formula to find the value of DM_IGM,integration[0] stores the result of
integration of function , func
    DM_e= DM_igm+ (108.66/(1+z[i]))
#Value of DM_HG_loc is fixed to 95.76
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum

#create an empty array k_igm to store the values of K_IGM (Range : 800 to
1100)
#create an empty array Omega_matter to store the values of Omega_matter
(Range : 0.001 to 0.500)
#create an empty array chi to store the chi square value corresponding to
each value of K_IGM and Omega_matter
#two parameter fitting using DM_IGM and Omega_matter taking DM_HG_loc
constant
k_igm=[]
Omega_matter=[]
chi=[]


for j in range(1,100):
  o_m= j*0.01
  for k in range(800,1100):
        chi_sq=find_chi_square(o_m,k)       #function call
        chi = np.append(chi,chi_sq)
        k_igm=np.append(k_igm,k)
        Omega_matter=np.append(Omega_matter,o_m)
chi_min=min(chi)
#find the minimum value in the chi array and store it in chi_min
index= np.where(chi==chi_min)
#find the index where the minimum value is stored in chi array
print(index)
print('k_igm_min:',k_igm[index],'\t','Chi_Square
Minimum:',chi_min,'\t','omega_m',Omega_matter[index])
#print the K_IGM and Omega_matter values corresponding to the value of chi
square minimum

d1=np.array([k_igm,Omega_matter]).T
#creating a numpy array of K_igm and Omega_matter
```

```
df=pd.DataFrame(data=d1, columns=["k_igm","Omega_matter"])
#create a dataframe using numpy array d1
df['chi_square']=chi        #add a column chi_square to the dataframe
display(df)

#to plot the contour plot between K_IGM and Omega_matter

plt.figure(figsize=(10,7))
for i in range(0,len(chi)):
  if (chi_min+2.3-0.4<=chi[i]<=chi_min+2.3+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="green",s=3)
#plot the points (colour:green) which lies between 1-sigma
  if (chi_min+6.17-0.4<=chi[i]<=chi_min+6.17+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="red",s=3)
#plot the points (colour:red) which lies between 2-sigma
  if (chi_min+11.8-0.4<=chi[i]<=chi_min+11.8+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="blue",s=3)
#plot the points (colour:blue) which lies between 3-sigma


# Plot
plt.scatter(k_igm[index],Omega_matter[index],color='black')
plt.xlim(0.2,0.42)    #define the range of x axis
plt.ylim(800,1000)    #define the range of y axis
plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel('K_IGM',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.show()

#function to find the chi square
def find_chi_square1(j,l):
  Sum=0
#Sum stores the values of chi_square for each value of DM_HG_loc and
Omega_matter
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(j*pow(1+z,3)+(1-j))**0.5
    integration=integrate.quad(func,0,z[i])
#integrate.quad function , integrate the given funnction, func from 0 to z
    #taking the value of k_igm as 921.16
```

```
    DM_igm=921.16* integration[0]
#formula to find the value of DM_IGM,integration[0] stores the result of integration of function , func
    DM_e= DM_igm+ (l/(1+z[i]))                #formula of DM_E
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum

#create an empty array DM_HG_loc to store the values of DM_HG_loc (Range : 50 to 160)
#create an empty array Omega_matter_1 to store the values of Omega_matter (Range : 0.001 to 0.500)
#create an empty array chi_1 to store the chi square value corresponding to each value of DM_HG_loc and Omega_matter
#two parameter fitting using DM_HG_loc and Omega_matter taking DM_IGM constant
DM_hg_loc=[]
Omega_matter_1=[]
chi_1=[]


for j in range(1,500):
  o_m_1= j*0.001
  for l in range(60,120):
        chi_sq_1=find_chi_square1(o_m_1,l)          #function call
        chi_1= np.append(chi_1,chi_sq_1)
#append the values chi_square into chi_1
        DM_hg_loc=np.append(DM_hg_loc,l)
#append the values DM_HG_loc into DM_HG_loc array
        Omega_matter_1=np.append(Omega_matter_1,o_m_1)
#append the value of Omega_matter into Omega_matter_1 array
chi_min_1=min(chi_1)
#find the minimum value in the chi_1 array and store it in chi_min
index_1= np.where(chi_1==chi_min_1)
#find the index where the minimum value is stored in chi array
print(index_1)
print('Chi_Square Minimum:',chi_min_1,'\t',chi_min_1,'\t','omega_m',Omega_matter_1[index_1],'\tDM_HG_loc',DM_hg_loc[index_1])
```

```
#print the DM_HG_loc and Omega_matter values corresponding to the value of
chi square minimum

d2=np.array([DM_hg_loc,Omega_matter_1]).T
#creating a numpy array of DM_HG_loc and Omega_matter
df2=pd.DataFrame(data=d2, columns=["DM_HG_loc","Omega_matter"])
#create a dataframe using numpy array d2
df2['chi_square']=chi_1       #add a column chi_square to the dataframe
display(df2)


plt.figure(figsize=(10,7))
#to plot the contour plot between DM_HG_loc and Omega_matter
for i in range(0,len(chi_1)):
  if (chi_min_1+2.3-0.4<=chi_1[i]<=chi_min_1+2.3+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="green",s=3)
#plot the points (colour:green) which lies between 1-sigma
  if (chi_min_1+6.17-0.4<=chi_1[i]<=chi_min_1+6.17+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="red",s=3)
#plot the points (colour:red) which lies between 2-sigma
  if (chi_min_1+11.8-0.4<=chi_1[i]<=chi_min_1+11.8+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="blue",s=3)
#plot the points (colour:blue) which lies between 3-sigma
  if (chi_min_1+6.17-0.4<=chi_1[i]<=chi_min_1+6.17+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="red",s=3)
  if (chi_min_1+11.8-0.4<=chi_1[i]<=chi_min_1+11.8+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="blue",s=3)


#Plot
plt.scatter(DM_hg_loc[index_1],Omega_matter_1[index_1],s=0.9)
plt.xlim(0.2,0.42)     #define the range of x axis
plt.ylim(50,130)       #define the range of y axis
plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel('DM_HG_loc',size='18')
```

```
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')

plt.show()

#function to find the chi square
def find_chi_square2(j,l):
  Sum=0
#Sum stores the values of chi_square for each value of DM_HG_loc and
K_IGM
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+(1-0.31))**0.5
    integration=integrate.quad(func,0,z[i])
#integrate.quad function , integrate the given funnction, func from 0 to z
    DM_igm=k* integration[0]
#formula to find the value of DM_IGM, integration[0] stores the result of
integration of function , func
    DM_e= DM_igm+ (l/(1+z[i]))                            #formula of DM_E
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum

# Create an empty array DM_HG_loc_1 to store the values of DM_HG_loc
(Range : 60 to 120)
# Create an empty array k_igm_1 to store the values of K_IGM (Range : 800
to 1100)
# Create an empty array chi_2 to store the chi square value corresponding
to each value of DM_HG_loc and k_igm
# Two parameter fitting using DM_HG_loc and k_igm taking Omega_matter
constant

DM_hg_loc_1=[]
k_igm_1=[]
chi_2=[]


for k in range(800,1100):
  for l in range(160,260):
        chi_sq_2=find_chi_square2(k,l)              # Function call
        chi_2= np.append(chi_2,chi_sq_2)
```

```
# Append the values chi_square into chi_2
        k_igm_1=np.append(k_igm_1,k)
# Aappend the values k_igm into k_igm_1 array
        DM_hg_loc_1=np.append(DM_hg_loc_1,l)
# Append the value of Omega_matter into DM_hg_loc_1 array

chi_min_2=min(chi_2)
# Find the minimum value in the chi_1 array and store it in chi_min
index_2= np.where(chi_2==chi_min_2)
# Find the index where the minimum value is stored in chi array
print(index_2)
print('Chi_Square
Minimum:',chi_min_2,'\t','K_IGM',k_igm_1[index_2],'\tDM_HG_loc',DM_hg_loc_
1[index_2])
# Print the DM_HG_loc and k_igm values corresponding to the value of chi
square minimum

d3=np.array([DM_hg_loc_1,k_igm_1]).T
# Creating a numpy array of DM_hg_loc and K_igm
df3=pd.DataFrame(data=d3, columns=["DM_HG_loc","K_IGM"])
# Create a dataframe using numpy array d3
df3['chi_square']=chi_2    # Add a column chi_square to the dataframe
display(df3)

plt.figure(figsize=(10,7))

# To plot the contour plot between DM_HG_loc and K_IGM
for i in range(0,len(chi_2)):
  if (chi_min_2+2.3-0.4<=chi_2[i]<=chi_min_2+2.3+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="green",s=3)
# Plot the points (colour:green) which lies between 1-sigma
  if (chi_min_2+6.17-0.4<=chi_2[i]<=chi_min_2+6.17+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="red",s=3)
# Plot the points (colour:red) which lies between 2-sigma
  if (chi_min_2+11.8-0.4<=chi_2[i]<=chi_min_2+11.8+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="blue",s=3)
# Plot the points (colour:blue) which lies between 3-sigma


#Plot
```

```
plt.scatter(DM_hg_loc_1[index_2],k_igm_1[index_2],color='black')
plt.xlim(800,1000)    # Define the range of x-axis
plt.ylim(100,300)     # Define the range of y-axis
plt.xlabel('K_IGM',size='18')
plt.ylabel('DM_HG_loc',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.show()
```

Appendix 3.2:

```
pip install pynverse    #install pynverse library which contains
inversefunc which is used to calculate the inverse function

#importing the necessary libraries
import numpy as np
from scipy import integrate
import math as m
from pynverse import inversefunc
import pandas as pd
import matplotlib.pyplot as plt

#Constants
H_0=67.7   #Hubble Constant
omega_m=0.31
omega_l =0.69   #fraction of the energy of the universe due to the
cosmological constant
omega_b=0.049   #current baryon mass density fraction of the universe
f_IGM= 0.83     #fraction of baryon mass in IGM
K_IGM=933
m_p=1.67*(10**-27)

#P(z)=z*exp(-z)     Probabilty Distribution Function for redshift z
PDF= lambda z : z* m.exp(-z)
CDF= lambda z: 1-m.exp(-z)*(z+1) #Cumulative Distribution Function

U=np.random.uniform(0,0.801,500) #Generate 500 data points between 0 and
0.8 uniformly

z=[]
```

```
for i in range(len(U)):
  inv_CDF = inversefunc(CDF,U[i])   #Calculate the inverse function
  z=np.append(z,inv_CDF)            #Append the values of z in array z[]

z.sort()    #to sort the z array

# Create three empty array DM_IGM,DM_HG_loc and DM_E for storing the
values of DM_IGM,DM_HG_loc and DM_E
DM_IGM=[]
DM_HG_loc=[]
DM_E=[]
function=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+0.69)**0.5
for i in range(0,500):
  Integration=integrate.quad(function,0,z[i])
#integrate.quad function , integrate the given funnction, func from 0 to z
  DM_IGM_f=K_IGM * Integration[0]
#formula to find the value of DM_IGM,integration[0] stores the result of
integration of function , func
  DM_IGM=np.append(DM_IGM,np.random.normal(DM_IGM_f,100))
#introduce a randomness by taking normal distribution N(DM_IGM,100 pc
cm^(-3)) in DM_IGM
  DM_HG_loc=np.append(DM_HG_loc,np.random.normal(200,50))
#take DM_HG_loc as normal distribution N(100 pc cm^(-3),20 pc cm^(-3))
  DM_E_formula=DM_IGM[i]+(DM_HG_loc[i]/(1+z[i]))
#formula to calculate value of DM_E
  DM_E=np.append(DM_E,DM_E_formula)

log_DM_E=np.log10(DM_E)     #np.log10 calculate the log of all the data
points in DM_E array and new array created is stored in log_DM_E
log_z=np.log10(z)           #np.log10 calculate the log of all the data
points in z array and new array created is stored in log_z

plt.scatter(log_z,log_DM_E,s=4) #Scatter plot between datapoints of
log_DM_E and log_z array
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples (DM_hg_loc=(200,50)).jpg")
```

```
poly_deg = 2
p = np.polyfit(log_z,log_DM_E, poly_deg)
print(p)
y_fit = np.poly1d(p) #use np.poly1d to give the polynomial function
corresponding to the coeffs. in p

#Plot the data and the fitted function:
plt.plot(log_z, y_fit(log_z), ls='-', label='polynomial of deg.
{}'.format(poly_deg),color='red')   #to plot the best fit line of degree 2

plt.scatter(log_z,log_DM_E,s=6)    #to plot scatter plot between log_z and
log_DM_E
plt.xlabel("log z")
plt.ylabel("log $DM_{E}$")
plt.title("log $DM_{E}$ versus log z")
plt.show()
plt.savefig("500 Samples (DM_hg_loc=(200,50)).jpg")

#function to find chi_square
def find_chi_square(j,k):
  Sum=0
#Sum stores the values of chi_square for each value of K_IGM and
Omega_matter
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(j*pow(1+z,3)+(1-j))**0.5
    integration=integrate.quad(func,0,z[i])
#integrate.quad function , integrate the given funnction, func from 0 to z
    DM_igm=k * integration[0]
#formula to find the value of DM_IGM,integration[0] stores the result of
integration of function , func
    DM_e= DM_igm+ (207.49/(1+z[i]))
# Value of DM_HG_loc is fixed to 95.76
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum

#create an empty array k_igm to store the values of K_IGM (Range : 800 to
1100)
#create an empty array Omega_matter to store the values of Omega_matter
(Range : 0.001 to 0.500)
```

```
#create an empty array chi to store the chi square value corresponding to
each value of K_IGM and Omega_matter
#two parameter fitting using DM_IGM and Omega_matter taking DM_HG_loc
constant
k_igm=[]
Omega_matter=[]
chi=[]


for j in range(1,100):
  o_m= j*0.01
  for k in range(800,1100):
        chi_sq=find_chi_square(o_m,k)     #function call
        chi = np.append(chi,chi_sq)
        k_igm=np.append(k_igm,k)
        Omega_matter=np.append(Omega_matter,o_m)
chi_min=min(chi)
#find the minimum value in the chi array and store it in chi_min
index= np.where(chi==chi_min)
#find the index where the minimum value is stored in chi array
print(index)
print('k_igm_min:',k_igm[index],'\t','Chi_Square
Minimum:',chi_min,'\t','omega_m',Omega_matter[index])
#print the K_IGM and Omega_matter values corresponding to the value of chi
square minimum

d1=np.array([k_igm,Omega_matter]).T
#creating a numpy array of K_igm and Omega_matter
df=pd.DataFrame(data=d1, columns=["k_igm","Omega_matter"])
#create a dataframe using numpy array d1
df['chi_square']=chi
#add a column chi_square to the dataframe
display(df)

#to plot the contour plot between K_IGM and Omega_matter

plt.figure(figsize=(10,7))
for i in range(0,len(chi)):
  if (chi_min+2.3-0.4<=chi[i]<=chi_min+2.3+0.4):
```

```
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="green",s=3)
#plot the points (colour:green) which lies between 1-sigma
  if (chi_min+6.17-0.4<=chi[i]<=chi_min+6.17+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="red",s=3)
#plot the points (colour:red) which lies between 2-sigma
  if (chi_min+11.8-0.4<=chi[i]<=chi_min+11.8+0.4):
    plt.scatter(df["Omega_matter"][i],df["k_igm"][i],color="blue",s=3)
#plot the points (colour:blue) which lies between 3-sigma


# Plot
plt.scatter(k_igm[index],Omega_matter[index],color='black')
plt.xlim(0.2,0.42)       #define the range of x axis
plt.ylim(800,1000)       #define the range of y axis
plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel('K_IGM',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.show()

#function to find the chi square
def find_chi_square1(j,l):
  Sum=0
#Sum stores the values of chi_square for each value of DM_HG_loc and
Omega_matter
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(j*pow(1+z,3)+(1-j))**0.5
    integration=integrate.quad(func,0,z[i])
#integrate.quad function , integrate the given funnction, func from 0 to z
    #taking the value of k_igm as 928.89
    DM_igm=928.89* integration[0]
#formula to find the value of DM_IGM,integration[0] stores the result of
integration of function , func
    DM_e= DM_igm+ (l/(1+z[i]))                          #formula of DM_E
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum

#create an empty array DM_HG_loc to store the values of DM_HG_loc (Range :
50 to 160)
```

```
#create an empty array Omega_matter_1 to store the values of Omega_matter
(Range : 0.001 to 0.500)
#create an empty array chi_1 to store the chi square value corresponding
to each value of DM_HG_loc and Omega_matter
#two parameter fitting using DM_HG_loc and Omega_matter taking DM_IGM
constant

DM_hg_loc=[]
Omega_matter_1=[]
chi_1=[]


for j in range(1,500):
  o_m_1= j*0.001
  for l in range(160,260):
        chi_sq_1=find_chi_square1(o_m_1,l)          #function call
        chi_1= np.append(chi_1,chi_sq_1)
#append the values chi_square into chi_1
        DM_hg_loc=np.append(DM_hg_loc,l)
#append the values DM_HG_loc into DM_HG_loc array
        Omega_matter_1=np.append(Omega_matter_1,o_m_1)
#append the value of Omega_matter into Omega_matter_1 array
chi_min_1=min(chi_1)
#find the minimum value in the chi_1 array and store it in chi_min
index_1= np.where(chi_1==chi_min_1)
#find the index where the minimum value is stored in chi array
print(index_1)
print('Chi_Square
Minimum:',chi_min_1,'\t',chi_min_1,'\t','omega_m',Omega_matter_1[index_1],
'\tDM_HG_loc',DM_hg_loc[index_1])
#print the DM_HG_loc and Omega_matter values corresponding to the value of
chi square minimum

d2=np.array([DM_hg_loc,Omega_matter_1]).T
#creating a numpy array of DM_HG_loc and Omega_matter
df2=pd.DataFrame(data=d2, columns=["DM_HG_loc","Omega_matter"])
#create a dataframe using numpy array d2
df2['chi_square']=chi_1
#add a column chi_square to the dataframe
display(df2)
```

```
#to plot the contour plot between DM_HG_loc and Omega_matter
err=0.1
plt.figure(figsize=(10,7))
for i in range(0,len(chi_1)):
  if (chi_min_1+2.3-0.4<=chi_1[i]<=chi_min_1+2.3+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="green",s=3)
#plot the points (colour:green) which lies between 1-sigma
  if (chi_min_1+6.17-0.4<=chi_1[i]<=chi_min_1+6.17+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="red",s=3)
#plot the points (colour:red) which lies between 2-sigma
  if (chi_min_1+11.8-0.4<=chi_1[i]<=chi_min_1+11.8+0.4):

plt.scatter(df2["Omega_matter"][i],df2["DM_HG_loc"][i],color="blue",s=3)
#plot the points (colour:blue) which lies between 3-sigma


#Plot
plt.scatter(DM_hg_loc[index_1],Omega_matter_1[index_1],color='black')
plt.xlim(0,0.42)         #define the range of x axis
plt.ylim(150,250)        #define the range of y axis
plt.xlabel(r'$\Omega_m$',size='18')
plt.ylabel('DM_HG_loc',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')

plt.show()

#function to find the chi square
def find_chi_square2(j,l):
  Sum=0
#Sum stores the values of chi_square for each value of DM_HG_loc and
K_IGM
  for i in range(500):
    func=lambda z:((7/8)*(1+z))/(0.31*pow(1+z,3)+(1-0.31))**0.5
    integration=integrate.quad(func,0,z[i])
#integrate.quad function , integrate the given funnction, func from 0 to z
```

```
    DM_igm=k* integration[0]
#formula to find the value of DM_IGM, integration[0] stores the result of
integration of function , func
    DM_e= DM_igm+ (l/(1+z[i]))
#formula of DM_E
    formula=pow((DM_E[i]- DM_e),2)/(pow(100,2) +pow((20/(1+z[i])),2))
    Sum=Sum+ formula
  return Sum

# Create an empty array DM_HG_loc_1 to store the values of DM_HG_loc
(Range : 60 to 120)
# Create an empty array k_igm_1 to store the values of K_IGM (Range : 800
to 1100)
# Create an empty array chi_2 to store the chi square value corresponding
to each value of DM_HG_loc and k_igm
# Two parameter fitting using DM_HG_loc and k_igm taking Omega_matter
constant

DM_hg_loc_1=[]
k_igm_1=[]
chi_2=[]


for k in range(800,1100):
  for l in range(160,260):
        chi_sq_2=find_chi_square2(k,l)            # Function call
        chi_2= np.append(chi_2,chi_sq_2)
# Append the values chi_square into chi_2
        k_igm_1=np.append(k_igm_1,k)
# Append the values k_igm into k_igm_1 array
        DM_hg_loc_1=np.append(DM_hg_loc_1,l)
# Append the value of Omega_matter into DM_hg_loc_1 array

chi_min_2=min(chi_2)
# Find the minimum value in the chi_1 array and store it in chi_min
index_2= np.where(chi_2==chi_min_2)
# Find the index where the minimum value is stored in chi array
print(index_2)
```

```
print('Chi_Square
Minimum:',chi_min_2,'\t','K_IGM',k_igm_1[index_2],'\tDM_HG_loc',DM_hg_loc_
1[index_2])
# Print the DM_HG_loc and k_igm values corresponding to the value of chi
square minimum

d3=np.array([DM_hg_loc_1,k_igm_1]).T
# Creating a numpy array of DM_hg_loc and K_igm
df3=pd.DataFrame(data=d3, columns=["DM_HG_loc","K_IGM"])
# Create a dataframe using numpy array d3
df3['chi_square']=chi_2
# Add a column chi_square to the dataframe
display(df3)

# To plot the contour plot between DM_HG_loc and K_IGM

plt.figure(figsize=(10,7))
for i in range(0,len(chi_2)):
  if (chi_min_2+2.3-0.4<=chi_2[i]<=chi_min_2+2.3+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="green",s=3)
# Plot the points (colour:green) which lies between 1-sigma
  if (chi_min_2+6.17-0.4<=chi_2[i]<=chi_min_2+6.17+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="red",s=3)
# Plot the points (colour:red) which lies between 2-sigma
  if (chi_min_2+11.8-0.4<=chi_2[i]<=chi_min_2+11.8+0.4):
    plt.scatter(df3["K_IGM"][i],df3["DM_HG_loc"][i],color="blue",s=3)
# Plot the points (colour:blue) which lies between 3-sigma



#Plot
plt.scatter(DM_hg_loc_1[index_2],k_igm_1[index_2],color='black')
plt.xlim(925,960)     # Define the range of x-axis
plt.ylim(100,300)      # Define the range of y-axis
plt.xlabel('K_IGM',size='18')
plt.ylabel('DM_HG_loc',size='18')
plt.title('Maximum Likelihood Estimation For Two Parameter
Fitting',color='purple',size='20')
plt.show()
```