MARKOV CHAIN MONTE CARLO METHODS : COSMOLOGICAL PARAMETER ESTIMATION

Dissertation submitted to the

Mahatma Gandhi University

In partial fulfillment of the requirements

For the Degree of

MASTER OF SCIENCE

in

PHYSICS

Author

ANTO I LONAPPAN

Reg. No: 21401671

Supervisor

Dr. CHARLES JOSE

Assistant Professor

DEPARTMENT OF PHYSICS

St Berchmans College

Founded 1922

AUTONOMOUS | College with Potential for Excellence | Reaccredited by NAAC with A Grade

CHANGANASSERY, KERALA

April 2016

St Berchmans College
Founded 1922
AUTONOMOUS | College with Potential for Excellence | Reaccredited by NAAC with A Grade
CHANGANASSERY, KERALA

DEPARTMENT OF PHYSICS

Certificate

Certified that this project report entitled *Markov Chain Monte Carlo Methods : Cosmological Parameter Estimation* submitted to Mahatma Gandhi University, Kottayam in partial fulfilment for the award of the degree of Master of Science in Physics, is a bonafide record of studies carried out by Anto.I.Lonappan, Reg.No: 21401671 during the year 2014-2016 at St. Berchmans College, Changanacherry and that it has not been previously submitted for the award of any degree or diploma or any other title of recognition.

Dr. Issac Paul
Head of the Department.

Dr. Charles Jose
Project Guide.

Dr. Jacob Mathew
Teacher-in-charge

Examiners : 1.

2.

Declaration

I, hereby declare that this dissertation entitled *Markov Chain Monte Carlo Methods : Cosmological Parameter Estimation* submitted to Mahatma Gandhi University, Kottayam in partial fulfillment of the requirement for the Degree of Master of Science in Physics, is a bonafide record of studies carried out by author during 2014-2016 at St. Berchmans College, Changanacherry, and that it has not been previously formed the basis for the award of any degree or diploma or any other title of recognition.In addition, all of the numerical work and all of the diagrams/graphs are the work of the author unless otherwise stated.

Anto I Lonappan

April 2016

Acknowledgment

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude to my Project Guide, Dr.Charles Jose for introducing me to this extremely interesting topic, for his thoughtful suggestions, insight and time for invaluable conversations.I would like to thank Dr. Jacob Mathew, my mentor, who has been very helpful and extremely supportive while the execution of this project. A vote of thanks to Dr.Issac Paul, HOD and faculties of Physics Department at SB College, changanacherry for their support and guidance. I am thankful to all my friends especially Thomas, for the valuable suggestions. Last but not least, I thank my parents for getting me started in this wonderful path of learning. This project is dedicated to my Father, who means the world to me.

Anto I Lonappan

Abstract

One of the main task in statistical analysis of cosmological models is the estimation of unknown cosmological parameters. In this project, we are interested in determining these unknown parameters especially in cases where the model contains a large number of parameters, using Markov Chain Monte Carlo (MCMC) methods. In paticular we have developed a multi thread MCMC algorithm developed in C++ which can be used for parallel applcations. We have tested the algorithm with several test problems. These problems include finding parameters of linear, parabola, quintic funtions and minimum of rosenbrock function. Gelman Rubin diagnostics is used for chains convergence test.We then used our algorithm to estimate cosmological parameteres for a flat ΛCDM universe using observations of Hubble constant at different redshifts [13][12][9][7][16] [17]. Best fit parameter values for curved universe are $\Omega_m = 0.29 \pm 0.012$, $\Omega_\Lambda = 0.69 \pm 0.033$ and H_0(km/s/Mpc) $= 71.23 \pm 1.43$. For a flat universe $\Omega_m = 0.28 \pm 0.012$ and H_0(km/s/Mpc)$= 70.98 \pm 0.95$. We have also obtained 2d marginalized distributions of these parameters.For this we have used python. Our results are consistent with other estimates from literature.

Literature Review

“ There is no result in nature without a cause; understand the cause and you will have no need for the experiment.”

— Leonardo da Vinci

Statistics is dominating in Cosmological studies, now a days more than ever:cosmological data from different experiments sets are getting ever larger.In that sense Cosmology had made transition from a data-starved science to a data-driven science,so sophisticated statistical tools are needed to study this science . Hence cosmology is intrinsically related to statistics, as theories of the origin and evolution of the Universe do not predict, for example, that a particular galaxy will form at a specific point in space and time or that a specific patch of the cosmic microwave background will have a given temperature: any theory will predict average statistical properties of our Universe, and we can only observe a particular realization of that.Cosmologists are interested in studying the origin and evolution of the physical Universe. They rely on data where a host of useful information is enclosed, but is encoded in a non-trivial way. The challenges in extracting this information must be overcome to make the most of the large experimental effort. Even after having analysed a decade or more of data and having converged to a standard cosmological model we should keep in mind that this model is described by 10 or more physical parameters and if we want to study deviations from the standard model the number of parameters is even larger. Dealing with such a high dimensional parameter space and finding parameters constraints is a challenge on itself. In addition, as gathering data is such an expensive and difficult process, cosmologists want to be able to compare and combine different data sets both for testing for possible disagreements and for improving parameter determinations. Finally, always because experiments are so expansive, cosmologists in many cases want to find out a priori, before actually doing the experiment, how much one would be able to learn from it. For all these reasons, more and more sophisticated statistical techniques are being employed in cosmology. The Markov chain Monte Carlo (MCMC) method, is one among such tool, has enjoyed an enormous upsurge in interest over the last few years. MCMC methods are widely used in many subjects like cosmology and astrophysics. It is useful in sampling probability density and calculating integrals especially when the models are expensive to compute. Many use of MCMC methods are based on modifications to the Metropolis-Hastings (M-H) algorithm.

Cosmological parametrisation is a method of estimating parameters of different cosmological models using statistical methods, where Markov chain Monte Carlo (MCMC) is a well sophisticated method.MCMC algorithm generates n-parameters in n-parameter space, thus the chain generation continues until required accuracy reaches, then how we know desired accuracy attained? there comes converging tests, best among those tests is Gelman and Rubin Multiple Sequence Diagnostic.Monte Carlo methods are a collection of techniques that use pseudo-random (computer simulated) values to approximate solutions to mathematical problems. The technique of Markov chain Monte Carlo (MCMC) is simple: if one wishes to sample randomly from a specific probability distribution then design a Markov chain whose long-time equilibrium is that distribution, write a computer program to simulate the Markov chain, run it for a time long enough to be confident that approximate equilibrium has been attained, then record the state of the Markov chain as an approximate draw from equilibrium. The simplicity of the underlying principle of MCMC is a major reason for its success. However a substantial complication arises a namely, how long should one run the Markov chain so as to ensure that it is close to equilibrium? This has stimulated much research, into mathematical upper bounds on rates of convergence, into varieties of different Markov chains for which convergence may be hoped to be rapid or susceptible to analysis.

The aim of MCMC is to generate a set of points in the parameter space whose distribution f unction is the same as the target density, in this case the likelihood, or more generally the posterior. MCMC makes random drawings, by moving in parameter space in a Markov process - i.e. the next sample depends on the present one, but not on previous ones. By design, the resulting Markov Chain of points samples the posterior, such that the density of points is proportional to the target density (at least asymptotically), so we can estimate all the usual quantities of interest from it (mean variance, etc). Markov Chain Monte Carlo refers to a class of algorithms for sampling from probability distributions in a special way. It does this by constructing a Markov Chain, which converges after a certain number of steps to the desired probability distribution. A Markov Chain is a process in which the next step or iteration of the process only depends upon the current step and not upon any previous steps in the process. If a Markov Chain has certain properties then the process will evolve in a random fashion until it reaches a certain state in which it remains thereafter, called equilibrium. Within the context of Bayesian Statistics, it is desirable to sample from a posterior distribution. An MCMC algorithm here constructs a Markov Chain in which the random evolution of the chain is probabilistic, each step in the chain constructs an empirical distribution, which is a Monte Carlo approximation of the posterior, and the chain converges to an equilibrium distribution, which is the posterior distribution. It is this fact regarding the guaranteed convergence to the desired equilibrium distribution, which enables the empirical distribution generated to serve as a Monte Carlo simulation of the posterior distribution.The Markov chain Monte Carlo (MCMC) method, as a computer-intensive statistical tool, has enjoyed an enormous upsurge in interest over the last few years In this project I am concentrating the discussion on the concept of free parameter estimation using a relatively new method called “Markov chain Monte Carlo method”(MCMC). In this report detail of study is organized into 7 chapters.

First chapter introduces the basic concepts of probabilities[8][10] and then Bayesian inference[15][14][5]. In our project we use normal distribution extensively, due to this multivariate normal distribution is also introduced Since we are playing with Bayesian inference its is required to figure out how much does theoretical model fits the data. So that point we use chi square test.

Second chapter is discussing about Monte Carlo methods and random number generation[5][6]. In this section, we provide the comparison of Pseudo random number generation (PRNG) technique and True Random number generation (TRNG) technique along with their applications.The third chapter deals with Markov chain Monte Carlo methods : How to produce Markov chain[1] using Metropolis Hasting algorithm[11].

Fourth chapter deals with implementation of these ideas to numerical computation[6] and concept of multi-thread program [4].Fifth chapter is about Algorithm testing and data analysis. Different testing equations and multi parameter polynomial are used for this purpose.Rosenbrock[6] function is also minimized using this algorithm.

Different tests done in chapter 5 and its results confirms the reliability of the program.In chapter six, basic concept of general relativity, metric and Friedmann equation is introduced. We selected Friedmann[3] ΛCDM model to be fitted. Last chapter contains the result, all available OHD[7][9][12][13][16][17] data points are collected and performed analysis.Concluded with the estimated parameter values.

List of Figures

List of Tables

Contents

Chapter 1

Probabilities

1.1 Probability Distributions

This chapter lists some of the major discrete and continuous probability distributions used in Monte Carlo simulation, along with specific algorithms for random variable generation.

1.2 Discrete Distributions

A discrete distribution describes the probability of occurrence of each value of a discrete random variable. A discrete random variable is a random variable that has countable values, such as a list of non-negative integers.With a discrete probability distribution, each possible value of the discrete random variable can be associated with a non-zero probability.We list various discrete distributions. Recall that a discrete distribution is completely specified by its discrete pdf.Bernoulli distribution ,Binomial distribution and so on.

1.2.1 Poisson Distribution

The pdf of the Poisson distribution is given by

$$f(x;\lambda) = \frac{\lambda_x}{x!}e^{-\lambda}, x = 0, 1, 2, ...,$$

where $\lambda > 0$ is the rate parameter. We write the distribution as $\mathrm{Poi}(\lambda)$. Examples of the graph of the pdf are given in Figure 1.1

The Poisson distribution is often used to model the number of arrivals of some sort during a fixed period of time. The Poisson distribution is closely related to the exponential distribution via the Poisson process

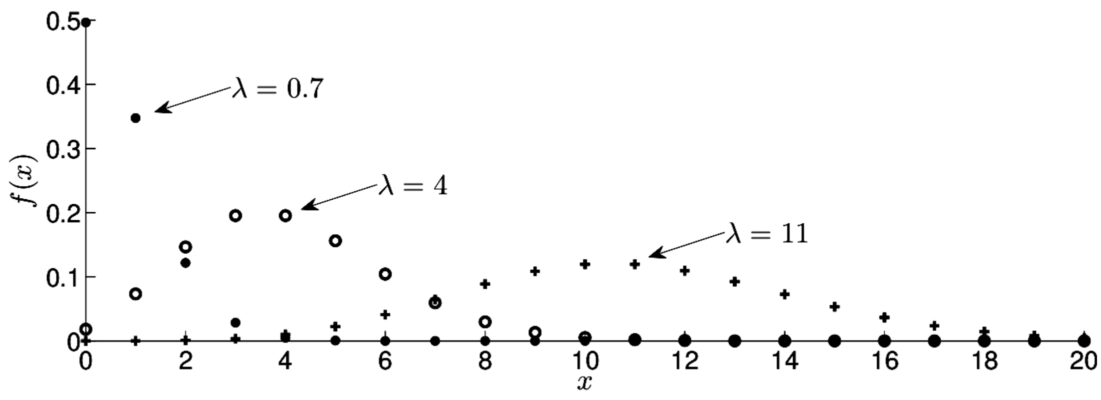

Figure 1.1: The pdfs of the Poi(0.7) (solid dot), Poi(4) (circle), and Poi(11)(plus) distributions.[8]

From this point of view the following is easy to prove: Let $Y_i \sim^{iid} Exp(\lambda)$.Then,

$$X = max\left\{n : \sum_{j=1}^{n} Y_j \leqslant 1\right\} \sim Poi(\lambda) \tag{1.1}$$

That is, the Poisson random variable X can be interpreted as the maximal number of iid exponential variables whose sum does not exceed 1. Let$U_i \sim^{iid} U(0,1)$. Rewriting (1.1), we see that

$$X = max\left\{n : \sum_{j=1}^{n} -lnU_j \leqslant \lambda\right\}$$

$$= max\left\{n : ln(\prod_{j=1}^{n} U_j) \geqslant -\lambda\right\}$$

$$= max\left\{n : \prod_{j=1}^{n} U_j \geqslant e^{-\lambda}\right\} \tag{1.2}$$

has a Poi(λ) distribution. This leads to the following algorithm

Algorithm -Poi(λ) Generator

1. Set n = 1 and a = 1.

2. Generate $U_n \sim U(0,1)$ and set a = a U_n .

3. If $a \leqslant e^{-\lambda}$, set n = n + 1 and go to Step 2.

4. Otherwise, return $X = n - 1$ as a random variable from Poi(λ).

For large λ alternative generation methods should be used.

1.2.2 Uniform Distribution

The discrete uniform distribution has pdf

$$f(x : a, b) = \frac{1}{b-a+1}, x \in a, , , , , b,$$

where $a, b \in \mathbb{Z}, b \geqslant a$ are parameters. The discrete uniform distribution is used as a model for choosing a random element from $a, ..., b$ such that each element is equally likely to be drawn. We denote this distribution by $DU(a, b)$.

Drawing from a discrete uniform distribution on $a, ..., b$, where a and b are integers, is carried out via a simple table lookup method.

Algorithm - DU(a, b) Generator

Draw $U \sim U(0, 1)$ *and output* $X = a + (b + 1 - a)U$.

1.3 Continuous Distributions

We list various continuous distributions in alphabetical order. Recall that an absolutely continuous distribution is completely specified by its pdf.

1.3.1 Normal Distribution

The standard normal or standard Gaussian distribution has pdf

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}, x \in \mathbb{R}.$$

The corresponding location–scale family of pdfs is therefore

$$f(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}, x \in \mathbb{R} \tag{1.3}$$

We write the distribution as$N(\mu, \sigma_2)$. We denote the pdf and cdf of the $N(0, 1)$ distribution as φ and Φ,respectively. Here, $\Phi(x) = \int_{-\infty}^{x} \varphi(t)dt = \frac{1}{2} + \frac{1}{2}erf(\frac{x}{\sqrt{2}})$, where erf(x) is the error function.

The normal distribution plays a central role in statistics and arises naturally as the limit of the sum of iid random variables via the central limit theorem. Its crucial property is that any affine combination of independent normal random variables is again normal. In Figure 1.2 the probability densities for three different normal distributions are depicted.

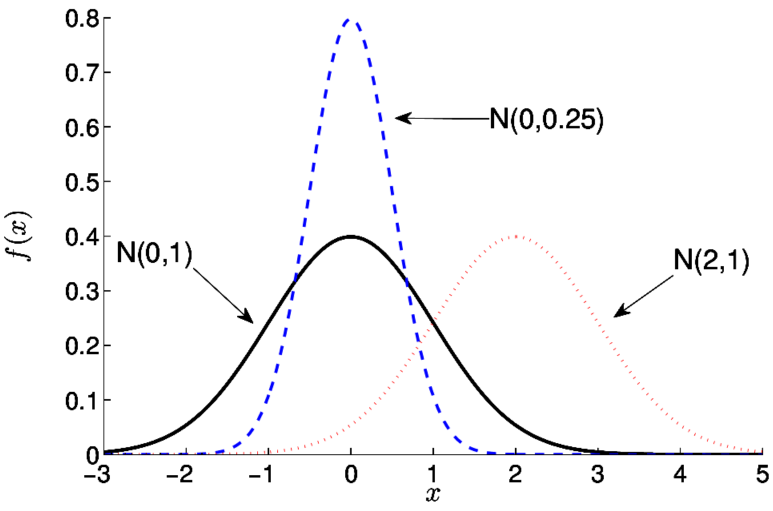

Figure 1.2: Pdfs of the normal distribution for various values of the parameters.[8]

Since $N(\mu, \sigma)$ forms a location–scale family, we only consider generation from $N(0, 1)$. The most prominent application of the polar method lies in the generation of standard normal random variables, leading to the celebrated Box-Muller method.

Algorithm (N(0, 1) Generator, Box-Muller Approach)

1. *Generate $U_1, U_2 \sim_{iid} U(0, 1)$.*
2. *Return two independent standard normal variables, X and Y, via*

$$X = \sqrt{-2lnU_1}\cos(2\pi U_2),$$

$$Y = \sqrt{-2lnU_1}\sin(2\pi U_2).$$

Finally, the following algorithm uses acceptance-rejection with an exponential proposal distribution. This gives a probability of acceptance of $\pi/(2e) \approx 0.76$

Algorithm (N(0, 1) Generator, Acceptance-Rejection

1. *Generate $X \sim Exp(1)$ and $U \sim U(0, 1)$, independently.*
2. *If $U' \leqslant e^{-(X-1)^2/2}$, generate $U \sim U(0, 1)$ and output $Z = (1 - 2I_{U\leqslant 1/2})X$; otherwise, repeat from Step 1.*

1.3.2 Uniform Distribution

The uniform distribution on the interval [a, b] has pdf

$$f(x; a, b) = \frac{1}{b-a}, a \leqslant x \leqslant b.$$

We write the distribution as U[a, b]. A graph of the pdf is given in Figure 1.3 The uniform

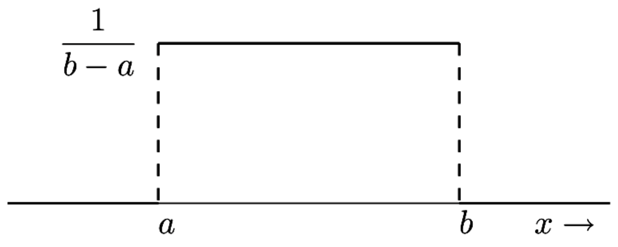

Figure 1.3: The pdf of the uniform distribution on [a, b].

distribution is used as a model for choosing a point randomly from the interval $[a, b]$, such that each point is equally likely to be drawn. The uniform distribution on an arbitrary Borel set B in $\mathbb{R}^n$ with non-zero Lebesgue measure (for example, area, volume) $|B|$ is defined similarly: its pdf is constant, taking value $1/|B|$ on B and 0 otherwise. We write U(B) or simply U. The U$[a, b]$ distribution is a location-scale family, as $Z \sim U[a, b]$ has the same distribution as $a + (b - a)X$, with $X \sim U[0, 1]$.U(a, b) random variable generation follows immediately from the inverse-transform method.

Algorithm (U(a, b) Generation)

Generate$U \sim U(0, 1)$ *and return* $X = a + (b - a)U$.

1.4 Multivariate Normal Distribution

The standard multivariate normal or standard multivariate Gaussian distribution in n dimensions has pdf

$$f(x) = \frac{1}{\sqrt{(2\pi)^n}} e^{\frac{1}{2}x^T x}, x \in \mathbb{R}^n \tag{1.4}$$

All marginals of $X = (X_1, ..., X_n)^T$ are iid standard normal random variables.

Suppose that$Z$ has an m-dimensional standard normal distribution. If A is an $n \times m$ matrix and μ is an $n \times 1$ vector, then the affine transformation

$$X = \mu + AZ$$

is said to have a multivariate normal or multivariate Gaussian distribution with mean vector μ and covariance matrix $\Sigma = AA^T$. We write the distribution as $N(\mu, \Sigma)$.

The covariance matrix Σis always symmetric and positive semi definite. When A is of full rank (that is, rank(A) $= minm, n$) the covariance matrix Σ is positive definite. In this case Σ has an inverse and the distribution of X has pdf

$$f(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n det(\Sigma)}} e^{\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}, x \in \mathbb{R}^n \tag{1.5}$$

The multivariate normal distribution is a natural extension of the normal distribution and plays a correspondingly important role in multivariate statistics. This multidimensional counterpart also has the property that any affine combination of independent multivariate normal random variables is again multivariate normal. A graph of the standard normal pdf for the two-dimensional case is given in Figure 1.4

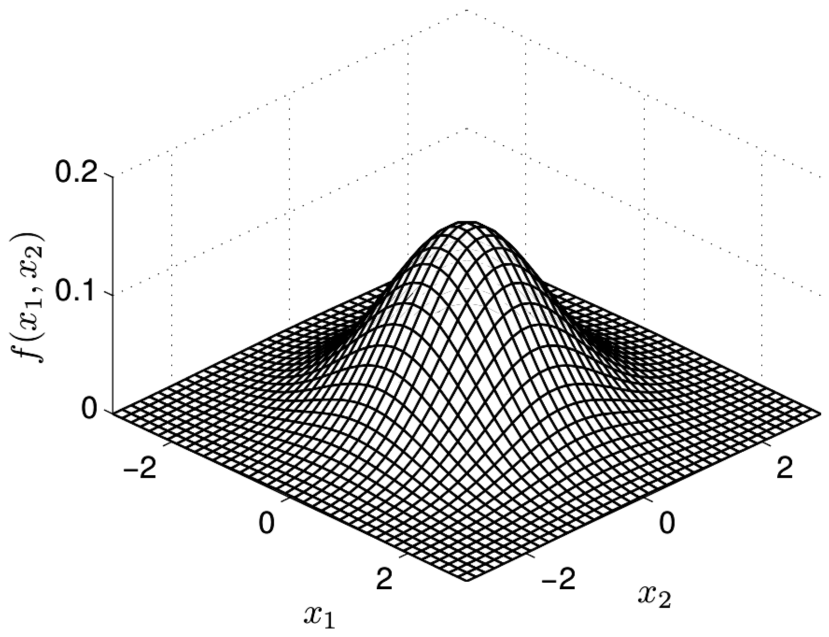

Figure 1.4: The standard multivariate normal pdf in two dimensions [8]

Some important properties are:

1. *Affine combinations*: Let $X_1, X_2, ..., X_r$ be independent m_i - dimensional normal variables, with$X_i \sim N(\mu_i, \sum_i), i = 1, ..., r$. Let a be an $n \times 1$ vector and let each A_i be an $n \times m_i$ matrix for $i = 1, ..., r$. Then,

$$a + \sum_{i=1}^{r} A_i X_i \sim N\left(a + \sum_{i=1}^{r} A_i \mu_i, \sum_{i=1}^{r} A_i \Sigma_i A_i^T\right).$$

In other words, any affine combination of independent multivariate normal random variables is again multivariate normal.

2. *Standardization (whitening)*: A particular case of the affine combinations property is the following. Suppose $X \sim N(\mu, \Sigma)$ is an n-dimensional normal random variable with $\det(\Sigma) > 0$. Let A be the Cholesky factor of the matrix Σ. That is, A is an$n \times n$ lower triangular matrix of the form

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \tag{1.6}$$

and such that $\Sigma = AA^T$. It follows that

$$A{-}1(X - \mu) \sim N(0, I).$$

3.*Marginal distributions*: Let X be an n-dimensional normal variable, with $X \sim N(\mu, \Sigma)$. Separate the vector X into a part of size p and one of size $q = n - p$ and, similarly, partition the mean vector and co-variance matrix:

$$X = \begin{pmatrix} X_p \\ X_q \end{pmatrix}, \; \mu = \begin{pmatrix} \mu_p \\ \mu_q \end{pmatrix}, \; \Sigma = \begin{pmatrix} \Sigma_p & \Sigma_r \\ \Sigma_r & \Sigma_q \end{pmatrix}, \tag{1.7}$$

where Σ_p is the upper left $p \times p$ corner of Σ, Σ_q is the lower right $q \times q$ corner o fΣ, andΣ_r is the $p \times q$ upper right block of Σ. Then, the distributions of the marginal vectors X_p and X_q are also multivariate normal, with $X_p \sim N(\mu_p, \Sigma_p)$ and $X_q \sim N(\mu_q, \Sigma_q)$.

Note that an arbitrary selection of p and q elements can be achieved by first performing a linear transformation $Z = AX$, where A is the $n \times n$ permutation matrix that appropriately rearranges the elements in X.

$$(X_p|X_q = x_q) \sim N(\mu_p + \Sigma_r\Sigma_q^{-1}(x_q - \mu_q), \; \Sigma_p - \Sigma_r\Sigma_q^{-1}\Sigma_r^T),$$

and

$$(X_q|X_p = x_p) \sim N(\mu_q + \Sigma_r^T\Sigma_p^{-1}(x_p - \mu_p), \; \Sigma_q - \Sigma_r^T\Sigma_p^{-1}\Sigma_r),$$

As with the marginals property, arbitrary conditioning can be achieved by first permuting the elements of X by way of a fine transformation using a permutation matrix.

Property 2 is the key to generating a multivariate normal random vector $X \sim N(\mu, \Sigma)$, and leads to the following algorithm.

Algorithm (N(μ, Σ) Generator)

1. Derive the Cholesky decomposition $\Sigma = AA^T$.

2. Generate $Z_1, ..., Z_n \sim^{iid} N(0,1)$ *and let* $Z = (Z_1, ..., Z_n)^T$.

3. Output $X = \mu + AZ$.

1.5 Bayesian Inference in Parameter Estimation

We collect some data, and wish to interpret them in terms of a model. A model[10] is a theoretical framework, which we assume, is true. The model consists of measurements Y, known quantities X, unknown parameters θ and measurement error ε. The problem is to estimate the unknown parameters θ based on the measurements Y. In this work, the problem is solved in the framework of Bayesian theory. That is, the error and the unknown parameters in the model are random variables and have a distribution - they are not thought to have a single ”correct” value, but different possible values, others being more probable than the others are. The goal of parameter estimation is to provide estimates of the parameters, and their errors, or ideally the whole probability distribution if given the data X. This is called the “ posterior probability distribution ”.

$$\wp(\theta, X)$$

From $\wp(\theta, X)$ one can[2] calculate the expectation values of the parameters, and their errors.

1.5.1 Bayesian vs. Frequentist Framework

In statistical analysis[5] there are two major approaches- the Frequentist and the Bayesian approach. In general, the goal in statistical inference is to make conclusions about a phenomenon based on observed data.In the Frequentist framework the observations made in the past are analysed with a created model and the result is regarded as confidence about the state of the real world. That is, we assume that the phenomenon modelled has statistical stability: the probabilities are defined as frequencies with which an event occurs if the experiment is run many times. An event with probability P is thought to occur nP times if the experiment is repeated n times.

In the Bayesian approach, the interpretation of probability is subjective. The belief quantified before is updated to present belief through new observed data. In the Bayesian framework the probability is never just a frequency (single value), but a distribution of possible values. In the previous example the frequency, nP can have different values of which some are more probable than the others - for every claim a probability can be assigned that tells how strong, our belief about the claim is. That is, the Bayesian inference is based on assigning degrees of beliefs for different events.

A common task in statistical analysis is the estimation of the unknown model parameters.The Frequentist approach relies on estimators derived from different data sets (experiments) and a specific sampling distribution of the estimators. In the Bayesian approach the solution encompasses various possible parameter values. Therefore, the Bayesian approach is by nature suitable for modelling uncertainty in the model parameters and model prediction. The Bayesian approach is based on prior and likelihood distributions of parameters The prior distribution includes our beliefs about the problem beforehand, whereas the likelihood represents the probabilities of observing a certain set of parameter values. The prior and the likelihood are updated to a posterior distribution, which represents the actual parameter distribution conditioned on the observed data, through the Bayesian rule.

1.5.2 Bayesian Rule

The Bayesian solution[14] to the parameter estimation task is the posterior distribution of the parameters, which is the conditional probability distribution of the unknown parameters given the observed data. That is, we are interested in the distribution with probability density function $P(\theta|X)$ .where θ denotes the unknown parameter values and X contains the observations. The posterior density can now be written in a form of the Bayesian Rule

$$P(\theta|X) = P(X|\theta)\frac{P(\theta)}{P(X)} \tag{1.8}$$

This is analogous to the Bayesian rule from the elementary probability calculus for two random variables A and B.

$$P(A/B) = \frac{P(B/A) \times P(A)}{P(B)} \tag{1.9}$$

<u>Proof</u>

$$P(A/B) = \frac{P(A \cup B)}{P(B)} \tag{1.10}$$

$$P(B/A) = \frac{P(A \cap B)}{P(A)} \tag{1.11}$$

Combining these two equations:

$$P(A|B) \times P(B) = P(B|A) \times P(A) \tag{1.12}$$

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(b)} \tag{1.13}$$

In equation (1.1):

$P(\theta/X)$ is the "posterior probability" for the parameters.

$P(X/\theta)$ is called the "Likelihood" and given its own symbol $L(X,\theta)$

$P(\theta)$ is called the " prior " and expresses what we know about the parameters prior to the experiment being done. This may be the result of previous experiments, or theory (e.g. some parameters, such as the age of the Universe, may have to be positive). In the absence of any previous information, the prior is often assumed a constant (a 'flat prior').

$P(X)$ is the evidence. For parameter estimation, the evidence simply acts to normalize the probabilities, i.e. $P(X) =_{-\infty} \int^{+\infty} P(X|\theta)P(\theta)d\theta$. In addition, the relative probabilities of the parameters do not depend on it, so it is often ignored and not even calculated. However, the evidence does play an important role in model selection, when more than one theoretical model is being considered, and one want to choose which model is most likely, whatever the parameters are. Actually all the probabilities above should be conditional probabilities, give any prior information I which we may have. It may be the result of previous experiments, or may be a theoretical prior, in the absence of any data. In such cases, it is common to adopt the principle of indifference and assume that all values of the parameter(s) is (are) equally likely, and take $P(\theta) = constant$ (perhaps within some finite bounds, or if infinite bounds, set it to some arbitrary constant and work with a non normalized prior). This is referred to as " flat prior ". Thus for flat prior, we have simply

$$P(\theta/X) \propto L(X;\theta)$$

Although we may have the full probability distribution for the parameters, often one simply uses the peak of the distribution as the estimate of the parameters. This is then a " Maximum Likelihood estimate ". Note that if the prior are not flat, the peak in the posterior$P(\theta/X)$ is not necessarily the maximum likelihood estimate. A ‘rule of thumb’ is that if the prior are assigned theoretically, and they influence the result significantly, the data are usually not good enough. (If the prior come from previous Experiment, the situation is different - we can be more certain that we really have some prior knowledge in this case). Finally, note that this method does not generally give a goodness of fit, only relative probabilities. It is still common to compute χ^2 at this point to check the fit is sensible.

1.5.3 Chi Square Test -Goodness of fit

Chi square distribution is a theoretical or mathematical distribution that has wide applicability in statistical work. The term ‘chi square’ (pronounced with a hard ‘chi’) is used because the Greek letter χ is used to denote this distribution. It will be seen that the elements on which this distribution is based are squared, so that the symbol χ^2 is used to denote the distribution. If v independent variables x_i are each normally distributed with mean μ and variance σ^2,then the quantity known as Chi-Square is defined by,

$$\chi^2 = \sum_{i=1}^{v} \frac{(X_i - \mu)^2}{\sigma_i^2}$$

Ideally, given the random fluctuations of values of X_i about their mean values μ_i, each term in sum will be of unity. If we chosen σ_i^2 and μ_i correctly, we expect that, the calculated value of Chi Square will approximately equal to V. Then, we conclude that data are well described by values we have chosen for μ_i . This is the general idea of chi Square Test. If you have a set of observations and have a model, described by a set of parameters q, and want to fit the model to the data. The model may be physically motivated or a convenient function. One then should define a merit function, quantifying the agreement between the model and the data, by maximizing the agreement one obtains the best-fit parameters. Any useful fitting procedure should provide:

1) Best fit parameters

2) Estimate of error on the parameters.

3) Possibly a measure of the goodness of fit.

One should bear in mind that if the model is a poor fit to the data then the recovered best-fit parameters are meaningless. Here we introduce the concept of model fitting (parameter fitting) using least squares. Let us assume we have a set of data points D_i, and a model for these data, $Y(X, q)$ which depends on set of parameters q. The least square, in its simplest incarnation is:

$$\chi^2 = \sum_i W_i[D_i - Y(X_i|\theta)]^2 \tag{1.14}$$

Where W_i are suitably defined weights. It s possible to show that the minimum variance weight is $W_i = 1/S_i^2$. Where S_i denote the error on data point i. In this case then the least squares is called “ Chi-square ”. The best fit parameters are those that minimize the χ^2.For a wide range of cases the probability distribution for different values of χ^2 around the minimum of Eq. (b)is the χ^2 distribution for $n = n - m$ degrees of freedom. Where n is the number of independent data points and m the number of parameters. The quantity Q evaluated the minimum chi-square (i.e. at the best-fit values for the parameters) give a measure of the goodness of fit.

⇒If Q gives a very small probability then there are three possible explanations:
1) The model is wrong and can be rejected.
2) The errors are underestimated.
3) The measurement errors are not Gaussian distributed.
⇒ If Q is too large then:
1) Errors have been overestimated
2) Data are correlated or non-independent
3) The distribution is non-Gaussian

This last case is very rare. A useful “ chi-by-eye ” rule is: the minimum χ^2 should be roughly equal to n(number of data - number of parameters). This is increasingly true for large n. From this, it is easy to understand the use of the so-called “ reduced chi-square ” that is the min/m: if m»n (i.e., number of data much larger than the number of parameters to fit, which should be true in the majority of the cases) then $m \approx n$ and the rule of thumb is that reduced chi-square should be unity. Note that the chi-square method, and the Q statistic, gives the probability for the data given a model $P(D|\theta)$, not $P(\theta|D)$.

Before moving into MCMC methods in parameter estimation, we take a closer look on the role of prior and proposal distributions from the point of view of parameter estimation.

1.5.4 Prior Distributions

he prior distribution describes our previous knowledge about the unknown parameters in the model. With properly selecting the prior distribution we can emphasize the parameters that we know to be more probable than the others. If we do not have any prior knowledge about the parameters, an uninformative prior can be used. That is, we state $P(\theta) = 1$. For informative prior it is often useful to use so called conjugate prior. This means that both the prior and the posterior come from the same family of distributions. Conjugate prior can be found, for example, for exponential and Gaussian (normal) distributions.

1.5.5 Proposal Distribution

The performance of the MH algorithm is dependent on how we choose the proposal distribution $q(\theta|\theta_{old})$. The proposal distribution has a large effect on the mixing of the chain, which means how well the samples are spread over the parameter space and its important parts. It is clear that with too wide a proposal distribution, many of the candidate points are rejected and the chain ”stays still” for long periods and the target distribution is reached slowly. Then again, when the proposal distribution is too narrow, the acceptance ratio is high but a representative sample of the target distribution is achieved slowly. There are two basic ways of constructing the proposal distribution. In the first approach we choose a fixed proposal $q(\theta|\theta_{old}) = q(\theta)$ that is independent of the previous state(parameter values). This is called Independent Metropolis-Hastings. A more practical approach considers the previously simulated value when the proposal is formed. That is, we perform a local search for new candidate points at the neighbourhood of the current value. This is why this algorithm, called the Random Walk Metropolis algorithm, is in some sources referred to as Local Metropolis, whereas the previous approach sometimes goes with the name Global Metropolis. The Random Walk approach is less dependent on the form of the proposal distribution in relation to the form of the target distribution. It is also more practical since we can use a standard, easy-to-sample proposal distribution, for example a Gaussian distribution centred at the current point. The Random Walk Metropolis-Hastings algorithm with a Gaussian proposal distribution is presented below:

Chapter 2

Introduction to Monte Carlo Methods

In very broad terms one can say that a computer simulation is the process of designing a model of a real or abstract system and then conducting numerical experiments using the computer to obtain a statistical understanding of the system behaviour. That is, sampling experiments are performed upon the model. This requires that certain variables in the model be assign random values associated with certain probability distributions. This sampling from various probability distributions requires the use of random numbers to create a stochastic simulation of the system behaviour. This stochastic simulation of system behaviour is called a Monte Carlo simulation. Monte Carlo simulations are used to construct theories for observed behaviour of complicated systems, predict future behaviour of a system, and study effects on final results based upon input and parameter changes within a system. The stochastic simulation is a way of experimenting with a system to find ways to improve or better understand the system behaviour.

Monte Carlo methods use the computer together with the generation of random numbers and mathematical models to generate statistical results that can be used to simulate and experiment with the behaviour of various business, engineering and scientific systems. Some examples of application areas where Monte Carlo modelling and testing have been used are: the simulation and study of analysis of mass production techniques, analysis of complex system behaviour, the study of shielding effects due to radiation, the modelling of atomic and subatomic processes, and the study of nuclear reactor behaviour. These are just a few of the numerous applications of Monte Carlo techniques. Monte Carlo simulations usually employ the application of random numbers which are uniformly distributed over the interval [0, 1]. These uniformly distributed random numbers are used for the generation of stochastic variables 412 from various probability distributions. These stochastic variables can then be used to approximate the behaviour of important system variables. In this way one can generate sampling experiments associated with the modelling of system behaviour. The statistician can then apply statistical techniques to analyse the data collected on system performance and behaviour over a period of time. The generation of a system variable behaviour from a specified probability distribution involves the use of uniformly distributed random numbers over the interval [0, 1] . The quantity of these random numbers generated during a Monte Carlo simulation can range anywhere from hundreds to hundreds of thousands. Consequently, the computer time necessary to run a Monte Carlo simulation can take anywhere from minutes to months depending upon the both the computer system and the application being simulated. The Monte Carlo simulation produces various numerical data associated with both the system performance and the variables affecting the system behaviour. These system variables which model the system behaviour are referred to as model parameters. The study of the sensitivity of model parameters and their affect on system performance is a large application area of Monte Carlo simulations. These type of studies involve a great deal of computer time and can be costly. We begin this introduction to Monte Carlo techniques with a discussion of random number generators.

2.1 Random Numbers

At the heart of any Monte Carlo method is a random number generator: a procedure that produces an infinite stream

$$U1, U2, U3, .. \approx (iid).distibution$$

of random variables that are independent and identically distributed (iid) according to some probability distribution, When this distribution is the uniform distribution on the interval (0,1) (that is, Dist = U(0, 1)), the generator is said to be a uniform random number generator. Most computer languages already contain a built-in uniform random number generator. The user is typically requested only to input an initial number, called the seed, and upon invocation the random number generator produces a sequence of independent uniform random variables on the interval (0, 1). In M MATLAB , for example,this is provided by the rand function.The concept of an infinite iid sequence of random variables is a mathematical abstraction that may be impossible to implement on a computer. The best one can hope to achieve in practice is to produce a sequence of “ random ” numbers with statistical properties that are indistinguishable from those of a true sequence of iid random variables. Although physical generation methods based on universal background radiation or quantum mechanics seem to offer a stable source of such true randomness, the vast majority of current random number generators are based on simple algorithms that can be easily implemented on a computer.

2.1.1 Properties of a Good Random Number Generator

What constitutes a good random number generator depends on many factors. It is always advisable to have a variety of random number generators available, as different applications may require different properties of the random generator. Below are some desirable, or indeed essential, properties of a good uniform random number generator:

1.*Pass statistical tests:* Pass statistical tests: The ultimate goal is that the generator should produce a stream of uniform random numbers that is indistinguishable from a genuine uniform iid sequence. Although from a theoretical point of view this criterion is too imprecise and even infeasible, from a practical point of view this means that the generator should pass a battery of simple statistical tests designed to detect deviations from uniformity and independence.

2. *Theoretical support:* A good generator should be based on sound mathematical principles, allowing for a rigorous analysis of essential proper- ties of the generator

3. *Reproducible:* An important property is that the stream of random numbers is reproducible without having to store the complete stream in memory. This is essential for testing and variance reduction techniques. Physical generation methods cannot be repeated unless the entire stream is recorded.

4. *Fast and efficient:* The generator should produce random numbers in a fast and efficient manner, and require little storage in computer memory. Many Monte Carlo techniques for optimization and estimation require billions or more random numbers. Current physical generation methods are no match for simple algorithmic generators in terms of

speed.

5. *Large period:* The period of a random number generator should be extremely large — on the order of 10 50 — in order to avoid problems with duplication and dependence. Most early algorithmic random number generators were fundamentally inadequate in this respect.

6.*Multiple streams:* In many applications it is necessary to run multiple in- dependent random streams in parallel. A good random number generator should have easy provisions for multiple independent streams.

7.*Cheap and easy:* A good random number generator should be cheap and not require expensive external equipment. In addition, it should be easy to install, implement, and run. In general such a random number generator is also more easily portable over different computer platforms and architectures.

8. *Not produce 0 or 1*: A desirable property of a random number generator is that both 0 and 1 are excluded from the sequence of random numbers. This is to avoid division by 0 or other numerical complications.

2.2 Random number Generation

Random numbers are sets of digits arranged in random order. Because they are randomly ordered, no individual digit can be predicted from knowledge of any other digit or group of digits. A random number generation is a process that produces random numbers. Any random process (e.g., a flip of a coin or the toss of a die) can be used to generate a Random Number. Random numbers are useful for a variety of purposes, such as generating data encryption keys, simulating and modelling complex phenomena and for selecting random samples from larger data sets. They have also been used aesthetically, for example in literature and music, and are of course ever popular for games and gambling. So simulations of random numbers are crucial. Things that are truly random are the measurement of physical phenomena such as thermal noises of semiconductor chips or radioactive sources.

A random number generator (RNG) is a computational or physical device designed to generate a sequence of numbers or symbols that can not be reasonably predicted better than by a random chance. Various applications of randomness have led to the development of several different methods for generating random data, of which some existed since ancient times, including dice, coin flipping, the shuffling of playing cards and many other techniques. Because of the mechanical nature of these techniques, generating large numbers of sufficiently random numbers (important in statistics) required a lot of work and/or time. Thus, results would sometimes be collected and distributed as random number tables. Nowadays, after the advent of computational random number generators, a growing number of government-run lotteries, and lottery games, are using RNGs instead of more traditional drawing methods.

Random number generators have applications in gambling, statistical sampling, computer simulation, cryptography, completely randomized design, and other areas where producing an unpredictable result is desirable. Generally, in applications having unpredictability as the paramount, such as in security applications, hardware generators are generally preferred over pseudo-random algorithms, where feasible. Random number generators are very useful in developing Monte Carlo-method simulations, as debugging is facilitated by the ability to run the same sequence of random numbers again by starting from the same random seed. They are also used in cryptography – so long as the seed is secret. Sender and receiver can generate the same set of numbers automatically to use as keys.

The generation of pseudo-random numbers is an important and common task in computer programming. While cryptography and certain numerical algorithms require a very high degree of apparent randomness, many other operations only need a modest amount of unpredictability. To generate a “ true ” random number, the computer measures some type of physical phenomenon that takes place outside of the computer. For example, the computer could measure the radioactive decay of an atom. According to quantum theory, there’s no way to know for sure when radioactive decay will occur, so this is essentially “ pure randomness ” from the universe. An attacker wouldn’t be able to predict when radioactive decay would occur, so they wouldn’t know the random value i.e, The first method measures some physical phenomenon that is expected to be random and then compensates for possible biases in the measurement process. Example sources include measuring atmospheric noise, thermal noise, and other external electromagnetic and quantum phenomena. For example, cosmic background radiation or radioactive decay etc.. The second method uses computational algorithms that can produce long sequences of apparently random results, which are in fact completely determined by a shorter initial value, known as a seed value or key. As a result, the entire seemingly random sequence can be reproduced if the seed value is known. This type of random number generator is often called a pseudo random number generator. This type of generators typically does not rely on sources of naturally occurring entropy, though it may be periodically seeded by natural sources. This generator type is non-blocking, so they are not rate-limited by an external event, making large bulk reads a possibility.

2.2.1 Pseudo random number generator (PRNG)

As the word ‘ pseudo ’ suggests, pseudo-random numbers are not random in the way you might expect, at least not if you’re used to dice rolls or lottery tickets. Essentially, PRNGs are algorithms that use mathematical formulae or simply pre calculated tables to produce sequences of numbers that appear random. A good example of a PRNG linear congruential method. A good deal of research has gone into pseudo-random number theory, and modern algorithms for generating pseudo-random numbers are so good that the numbers look exactly like they were really random.

The basic difference between PRNGs and TRNGs is easy to understand if you compare computer-generated random numbers to rolls of a die. Because PRNGs generate random numbers by using mathematical formulae or pre-calculated lists, using one corresponds to someone rolling a die many times and writing down the results. Whenever you ask for a die roll, you get the next on the list. Effectively, the numbers appear random, but they are really predetermined. TRNGs work by getting a computer to actually roll the die — or, more commonly, use some other physical phenomenon that is easier to connect to a computer than a die is. PRNGs are efficient, meaning they can produce many numbers in a short time, and deterministic, meaning that a given sequence of numbers can be reproduced at a later date if the starting point in the sequence is known. Efficiency is a nice characteristic if your application needs many numbers, and determinism is handy if you need to replay the same sequence of numbers again at a later stage. PRNGs are typically also periodic, which means that the sequence will eventually repeat itself. While periodicity is hardly ever a desirable characteristic, modern PRNGs have a period that is so long that it can be ignored for most practical purposes. These characteristics make PRNGs suitable for applications where many numbers are required and where it is useful that the same sequence can be replayed easily. Popular examples of such applications are simulation and modelling applications. PRNGs are not suitable for applications where it is important that the numbers are really unpredictable, such as data encryption and gambling.

2.2.2 True Random Number Generators (TRNG)

In comparison with PRNGs, TRNGs extract randomness from physical phenomena and introduce it into a computer. You can imagine this as a die connected to a computer, but typically people use a physical phenomenon that is easier to connect to a computer than a die is. The physical phenomenon can be very simple, like the little variations in somebody's mouse movements or in the amount of time between keystrokes. In practice, however, you have to be careful about which source you choose. For example, it can be tricky to use keystrokes in this fashion, because keystrokes are often buffered by the computer's operating system, meaning that several keystrokes are collected before they are sent to the program waiting for them. To a program waiting for the keystrokes, it will seem as though the keys were pressed almost simultaneously, and there may not be a lot of randomness there after all. . The characteristics of TRNGs are quite different from PRNGs. First, TRNGs are generally rather inefficient compared to PRNGs, taking considerably longer time to produce numbers. They are also non-deterministic, meaning that a given sequence of numbers cannot be reproduced, although the same sequence may of course occur several times by chance. TRNGs have no period.

Chapter 3

Markov Chain Monte Carlo Methods

Many situations arise in Bayesian Statistics in which the posterior distribution is difficult to either calculate or simulate. This owes much to the necessity to calculate integrals of complex and commonly multi-dimensional expressions. Even if a posterior distribution can be found up to a constant of proportionality, the presence of a large number of parameters in a Bayesian framework means that most methods of simulating random variables from this distribution break down. For many years such problems were either too difficult or too computationally intensive to be tackled. The discovery of Markov Chain Monte Carlo (MCMC) methods marked a new approach to these problems. The ability of MCMC to provide insight into large, complex Bayesian problems has been one of the most important developments in modern statistics and without which Bayesian Statistics would not be so popular today.

Markov Chain Monte Carlo refers to a class of algorithms for sampling from probability distributions in a special way. It does this by constructing a Markov Chain that converges after a certain number of steps to the desired probability distribution. A Markov Chain is a process in which the next step or iteration of the process only depends upon the current step and not upon any previous steps in the process. If a Markov Chain has certain properties then the process will evolve in a random fashion until it reaches a certain state in which it remains thereafter, called equilibrium. Within the context of Bayesian Statistics, it is desirable to sample from a posterior distribution. An MCMC algorithm here constructs a Markov Chain in which the random evolution of the chain is probabilistic, each step in the chain constructs an empirical distribution that is a Monte Carlo approximation of the posterior, and the chain converges to an equilibrium distribution that is the posterior distribution. It is this fact regarding the guaranteed convergence to the desired equilibrium distribution that enables the empirical distribution generated to serve as a Monte Carlo simulation of the posterior distribution.

The simplest MCMC algorithms are random walks which means that each step in the algorithm takes small, random steps around the target equilibrium (or posterior in a Bayesian context) distribution. After a sufficient number of steps have been taken then the full equilibrium distribution has been explored. A key issue is how these steps are defined and how to minimize the number of steps required exploring the entire target equilibrium distribution. There are no specific correct answers to these questions, but a range of different algorithms exist each offering a different way of sampling from the desired probability distribution. Some MCMC algorithms are tuned in order to fit the context of the problem at hand, since poor tuning can lead to poor Monte Carlo simulations.

There is a difference between MCMC and simple Monte Carlo simulation in that simple Monte Carlo simulation simulates independent random values from the probability distribution of interest. Clearly, by its very nature there is dependence between simulated values in MCMC methods. This dependence causes two main issues, which need to be taken into consideration when assessing the resulting Monte Carlo approximations from an MCMC algorithm. The first issue is that convergence of the chain can take a long time and even then, it is difficult to identify and prove that the chain has indeed converged. Convergence of the chain will improve the Monte Carlo approximations of the posterior distribution. The second issue regards the speed at which the chain explores the target equilibrium distribution. This is referred to as mixing. It is desirable to have rapid mixing and so therefore have fast convergence. These issues can be handled by careful consideration of the problem and effective application of appropriate MCMC algorithms. However, such issues are objective and so it is always to the discretion of the statistician of how to proceed.

The first major MCMC algorithm was devised by Metropolis *et al* (1953), the so-called Random Walk Metropolis algorithm. This algorithm constructs a Markov Chain by proposing small probabilistic symmetric “jumps” (the chain moves from one value to another in one-step) centered on the current state of the chain. These are either accepted or rejected according to some specified probability, and in the case of rejection, the next step in the chain equals the previous step. This algorithm was later generalized by Hastings (1970) to become the Metropolis-Hastings algorithm. Much the same as the Random Walk Metropolis algorithm, the Metropolis-Hastings also proposes probabilistic jumps dependent upon the current state of the chain. In contrast, these jumps do not have to be symmetric and indeed the probability of acceptance of one of these new values is different. The Random Walk Metropolis algorithm is a special case of the general Metropolis-Hastings algorithm. In addition, there are a few other widely used special cases of the Metropolis-Hastings algorithm: the Independence Sampler and the Gibbs Sampler algorithms. The Gibbs Sampler algorithm is very useful given that the algorithm does not need to be tuned. In order to apply the Gibbs Sampler the posterior distribution of many parameters must be able to be written as a product of conditional probability distributions of fewer parameters. A Markov Chain is then constructed by simulating each parameter in turn from the conditional distributions and there is no need to accept or reject each simulated value. However, often-posterior distributions cannot be written in the required form or indeed simulation from specific conditional distributions may be difficult.

3.1 Markov Chain Monte Carlo Methods

Markov chain Monte Carlo (MCMC) is a generic method for approxi- mate sampling from an arbitrary distribution. The main idea is to generate a Markov chain whose limiting distribution is equal to the desired distribution.

The aim of MCMC is to generate a set of points in the parameter space whose distribution function is the same as the target density, in this case the likelihood, or more generally the posterior. MCMC makes random drawings, by moving in parameter space in a Markov process - i.e. the next sample depends on the present one, but not on previous ones. By design, the resulting Markov Chain of points samples the posterior, such that the density of points is proportional to the target density (at least asymptotically), so we can estimate all the usual quantities of interest from it (mean, variance, etc.). In cosmology, we are often dealing with around 10-20 parameters, so MCMC has been found to be a very effective tool.

MCMC is particularly used in connection with Bayesian statistics. In order to generate points where a proposal distribution function $q(\theta;\theta_o)$,which can be any proposal distribution function from which independent random values θ can be generated, and which contains as a parameter another point in the same space θ_0.For example a Gaussian distribution centred at θ_0 can be used.

3.1.1 Markov Process: (chance or random process)

A Markov (memory less) process is a process whose entire past history is summarized in its current (present state) i.e., future is independent of the past .A consecutive set of random (not deterministic) quantities defined on some known state space Θ. Think of Θ as our parameter space. Consecutive implies a time component, indexed by t. Consider a draw of (t) to be a state at iteration t. The next draw $\theta(t+1)$ is dependent only on the current draw $\theta(t)$, and not on any past draws. This satisfies the " Markov property ": i.e.,

$$P(\theta(t+1)|\theta(1),\theta(2),...,\theta(t)) = P((t+1)|\theta(t)$$

3.1.2 Markov chain

The idea behind the MCMC[1] methods is to create a certain type of Markov Chain that represents the posterior distribution. Markov Process is a certain type of discrete time stochastic process. A Markov Chain is a series of states created by a Markov Process. Assume that we have a series of random variables, $(X(0), X(1)...)$. This series is a Markov Chain (produced by a Markov Process), if the value of $X(t+1)$ only depends on the value of the previous state $X(t)$. i.e., $P(\theta(t+1)|\theta(1),\theta(2),...,\theta(t)) = P(\theta(t+1)|\theta(t)$.Let us look at a Markov Process $X(t), t = 0, 1, 2$, that has a finite state space (for simplicity) S (say k possible states) and the states assume values from Rd. i.e., $S = s_1, s_2...s_k$. A state space here means the set of all possible values obtained by the process. So, our Markov chain is a bunch of draws of θ that are each slightly dependent on the previous one. The chain wanders around the parameter space, remembering only where it has been in the last period.

3.2 Metropolis–Hastings Algorithm

The MCMC method originates from Metropolis et al. and applies to the following setting. Suppose that we wish to generate samples from an arbitrary multidimensional pdf

$$f(x) = \frac{p(x)}{Z}, x \in X$$

,where p(x) is a known positive function and Z is a known or unknown normalizing constant. Let q(y — x) be a proposal or instrumental density: a Markov transition density describing how to go from state x to y. Similar to the acceptance–rejection method, the Metropolis–Hastings algorithm is based on the following " trial-and-error " strategy.

Algorithm 1 (Metropolis–Hastings Algorithm)

To sample from a density $f(x)$ known up to a normalizing constant, initialize with some X_0 for which $f(X_0) > 0$. Then, for each $t = 0, 1, 2, ..., T - 1$ execute the following steps

1. Given the current state X_t , generate $Y \sim q(y|X_t)$.

2. Generate $U \sim U(0, 1)$ and deliver

$$X_{t+1} = \begin{cases} Y & \text{if U} \leq \alpha(X_t, Y) \\ X_t & \text{otherwise} \end{cases} \tag{3.1}$$

where,

$$\alpha(x, y) = min\left\{\frac{f(y)q(x|y)}{f(x)q(y|x)}, 1\right\} \tag{3.2}$$

The probability $\alpha(x, y)$ is called the acceptance probability. Note that in(3.2) we may replace f by p.We thus obtain the so-called Metropolis-Hastings Markov chain, $X_0, X_1, ..., X_T$, with X_T approximately distributed according to $f(x)$ for large T . A single Metropolis-Hastings iteration is equivalent to generating a point from the transition density $k(x_{t+1}|x_t)$, where

$$k(y|x) = \alpha(x, y)q(y|x) + (1 - \alpha_*(x))\delta_x(y), \tag{3.3}$$

with $\alpha_*(x) = \int \alpha(x, y)q(y|x)dy$ and $\delta_x(y)$ denoting the Dirac delta function.
Since,

$$f(x)\alpha(x, y)q(y|x) = f(y)\alpha(y, x)q(x|y)$$

and

$$(1 - \alpha_*(x))\delta_x(y)f(x) = (1 - \alpha_*(y))\delta y(x)f(y),$$

the transition density satisfies the detailed balance equation:

$$f(x)k(y|x) = f(y)k(x|y),$$

from which it follows that f is the stationary pdf of the chain. In addition, if the transition density q satisfies the conditions.

$$\mathbb{P}(\alpha(X_t, Y) < 1|X_t) > 0,$$

that is, the event $X_{t+1} = X_t$ has positive probability, and

$$q(y|x) > 0 \; \forall \; x, y \in \mathscr{X},$$

then f is the limiting pdf of the chain. As a consequence, to estimate an expectation $\mathbb{E}H(X)$, with $X \sim f$, one can use the following ergodic estimator:

$$\frac{1}{T+1}\sum_{t=0}^{T} H(x_t) \tag{3.4}$$

The original Metropolis algorithm is suggested for symmetric proposal functions; that is, for $q(y|x) = q(x|y)$. Hastings modified the original MCMC algorithm to allow non-symmetric proposal functions, hence the name Metropolis-Hastings algorithm.

3.2.1 Independence Sampler

If the proposal function $q(y|x)$ does not depend on x, that is,$q(y|x) = g(y)$ for some pdf (y), then the acceptance probability is

$$\alpha(x, y) = min\left\{\frac{f(y)q(x)}{f(x)g(y)}, 1\right\}$$

and Algorithm 1 is referred to as the independence sampler. The independence sampler is very similar to the acceptance-rejection method. Just as in that method, it is important that the proposal density g is close to the target f . Note, however, that in contrast to the acceptance-rejection method the independence sampler produces dependent samples. In addition, if there is a constant C such that

$$f(x) = \frac{p(x)}{\int p(x)dx} \leq Cg(x)$$

for all x, then the acceptance rate in (3.1) is at least 1/C whenever the chain is in stationary

3.2.2 Random Walk Sampler

If the proposal is symmetric, that is, q(y|x) = q(x|y), then the acceptance probability (3.2) is

$$\alpha(x, y) = min\left\{\frac{f(y)}{f(x)}, 1\right\} \tag{3.5}$$

and Algorithm 1 is referred to as the random walk sampler. An example of a random walk sampler is when $Y = Xt + \sigma Z$ in Step 1 of Algorithm 1, where Z is typically generated from some spherically symmetrical distribution (in the continuous case), such as $N(0, I)$.

3.2.3 Algorithm of MCMC

The MCMC employs a Markov chain random walk , whereby the new sample in parameter space, designated $Xt + 1$, depends on previous sample $X(t)$ according to an entity called the transition probability $P(X(t + 1)|X(t))$. The transition probability is assumed to be time independent.

1. Choosing X_0 as an initial location in the parameter space, the algorithm iterates the following steps.

2. Compute the probability $P(X_0)$.

3. Generate a value X_1 from proposal distribution $q(X_1/X_0)$.where $q(X_1/X_0)$ is a normal distribution with mean X_0.

4. Accepting it as a new point in the chain, which depends on the ratio of the new and old probabilities .The most popular algorithm, is the Metropolis-Hastings algorithm. Where the new value is accepted if the proposed X_1 is at a value of probability higher than X_0. i.e., if $[P(X_1)/P(X_0)] > 1$ the new point is accepted and process is repeated.

5. If $[P(X_1)/P(X_0)] < 1$: Generate a value Q uniformly distributed in [0, 1]

6 .If $[P(X_1)/P(X_0)] < Q$: then also the new point is accepted and the process is repeated.

7. If $[P(X_1)/P(X_0)] > Q$: then old point X_0 is repeated.

8. Finally, the chain converges to a point: which will give us the value we required.

3.2.4 Standard MCMC Flowchart

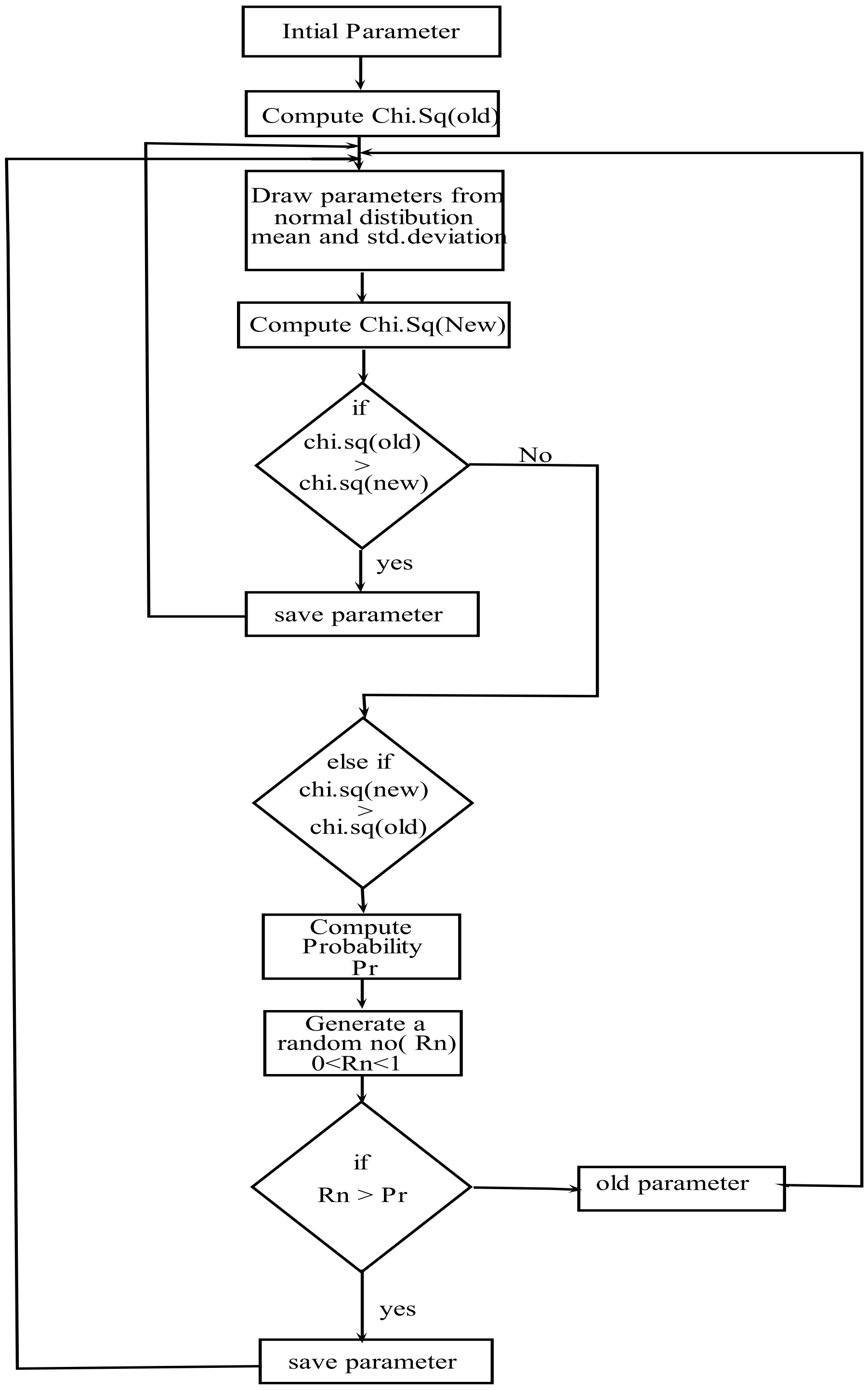
Intial Parameter
Compute Chi.Sq(old)
Draw parameters from normal distibution mean and std.deviation
Compute Chi.Sq(New)
if chi.sq(old) > chi.sq(new)
No
yes
save parameter
else if chi.sq(new) > chi.sq(old)
Compute Probability Pr
Generate a random no(Rn) 0<Rn<1
if Rn > Pr
old parameter
yes
save parameter

Figure 3.1: Flowchart of MCMC

Chains generated by the MCMC algorithm depends on number of iterations.Here first row is of 1500 iterations, second row is of 3000 and third, 4500 iterations.It is shown in Figure 3.2.

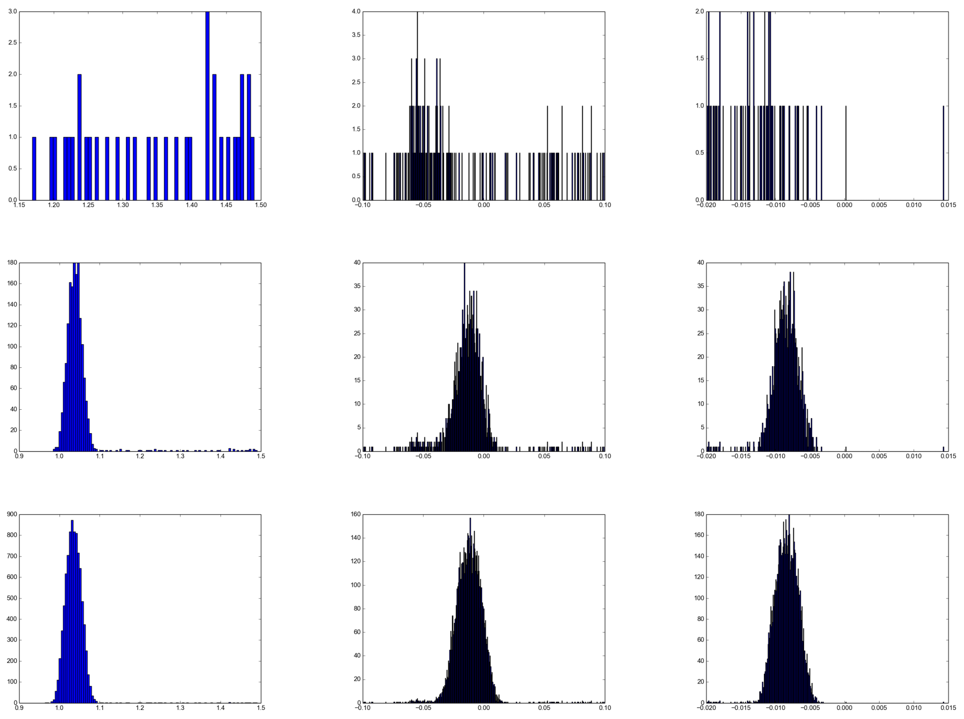

Figure 3.2: Gaussian Alarm : It shows that chains convergence depends on iterations.

3.3 Burn-in and convergence

Since convergence usually occurs regardless of our starting point, we can usually pick any feasible (for example, picking starting draws that are in the parameter space) starting point. However, the time it takes for the chain to converge varies depending on the starting point. As a matter of practice, most people throw out a certain number of the first draws, known as the " burn-in ". This is to make our draws closer to the stationary distribution and less dependent on the starting point. The length of the burn in period depends on the shape of the target distribution and the initial values θ_0. The period in which the chain has not yet converged must be discarded in order to avoid unrepresentative, "false" samples. During the burn-in period, the parameter values are not saved into the chain. How long should the burn-in period be? The simplest way is to run the MCMC algorithm a few times with different starting values and visually see where the parameter values converge to some equilibrium and thus base the length of the burn-in period on these observations. Theory indicates that the chain should fairly sample the target distribution once it has converged to a stationary distribution.

A difficult question in MCMC methods is whether the created chain is long enough so that it has reached its invariant distribution. Another issue that needs consideration is whether the algorithm has covered the target distribution sufficiently. How can we be sure that the Monte Carlo estimate of an integral converges to the right value as the number of samples generated approaches infinity? How fast is the method converging to the right value and how accurate the estimate approximately is when a certain number of samples are produced? The methods that address this issue belong to the field of MCMC convergence diagnostics. Convergence diagnostics here means statistical analysis done in order to asses convergence of the MCMC algorithm. The classic test is the " Gelman -Rubin (1992) convergence criterion ". Start M chains, each with 2N points, starting at well-separated parts of parameter space. The first N is discarded in burn-in. The idea is that we have two ways to estimate the mean of the parameters - either treat the combined chains as a single dataset, or look at the means of each chain. If the chains have converged, these should agree within some tolerance.

Let I represent the point in parameter space in position I of chain J. Compute the mean of each chain (J):

$$\overline{\theta} \equiv \frac{1}{N}\sum_{I=1}^{N}\theta_I^J$$

and the mean of all the chains

$$\overline{\theta} \equiv \frac{1}{NM}\sum_{I=1}^{N}\sum_{J=1}^{M}\theta_I^J$$

The chain-to-chain variance B is

$$B = \frac{1}{(M-1)}\sum_{J=1}^{M}(\overline{\theta}^J - \overline{\theta})^2$$

and the average variance of each chain is

$$W = \frac{1}{M(N-1)}\sum_{I=1}^{N}\sum_{J=1}^{M}(\overline{\theta}_I^J - \overline{\theta}^J)^2$$

Under convergence, W and B/N should agree. The weighted estimate of the variance,

$$\sigma^2 = \frac{(N-1)}{N}W + \frac{B}{N}$$

overestimates the true variance if the starting distribution is over dispersed. Accounting for the variance of the means gives an estimator of the variance

$$V = \sigma^2 + \frac{B}{NM}$$

V is an overestimate, and W is an underestimate. The ratio of the two estimates is

$$\hat{R} = \frac{\frac{(N-1)}{N}W + B(1 + \frac{1}{M})}{W}$$

R should approach unity as convergence is achieved. How close to R = 1? Opinions differ;I have seen suggestions to run the chain until the values of $\hat{R}$ are always < 1.03, or <1.02.

Chapter 4

Numerical and Computational Implementation

4.1 C++ programming language

C++ was developed by Bjarne Stroustrup of AT& T Bell Laboratories in the early 1980's, and is based on the C language. The"++" is a syntactic construct used in C (to increment a variable), and C++ is intended as an incremental improvement of C. Most of C is a subset of C++, so that most C programs can be compiled (i.e. converted into a series of low-level instructions that the computer can execute directly) using a C++ compiler.

C is in many ways hard to categorise. Compared to assembly language it is high-level, but it nevertheless includes many low-level facilities to directly manipulate the computer's memory. It is therefore an excellent language for writing efficient "systems" programs. But for other types of programs, C code can be hard to understand, and C programs can therefore be particularly prone to certain types of error. The extra object-oriented facilities in C++ are partly included to overcome these shortcomings.

The use of C++ has changed dramatically over the years and so has the language itself. From the point of view of a programmer, most of the changes have been improvements. The current ISO standard C++ (ISO/IEC 14882-2011, usually called C++11) is simply a far better tool for writing quality software than were previous versions. How is it a better tool? What kinds of programming styles and techniques does modern C++ support? What language and standard-library features support those techniques? What are the basic building blocks of elegant, correct, maintainable, and efficient C++ code? Those are the key questions answered by this book. Many answers are not the same as you would find with 1985, 1995, or 2005 vintage C++: progress happens.

C++ is a general-purpose programming language emphasizing the design and use of type-rich, lightweight abstractions. It is particularly suited for resource-constrained applications, such as those found in software infrastructures. C++ rewards the programmer who takes the time to master techniques for writing quality code. C++ is a language for someone who takes the task of programming seriously. Our civilization depends critically on software; it had better be quality software.

There are billions of lines of C++ deployed. This puts a premium on stability, so 1985 and 1995 C++ code still works and will continue to work for decades. However, for all applications, you can do better with modern C++; if you stick to older styles, you will be writing lower-quality and worse-performing code. The emphasis on stability also implies that standards-conforming code you write today will still work a couple of decades from now.

4.1.1 The Design of C++

The purpose of a programming language is to help express ideas in code. In that, a programming language performs two related tasks: it provides a vehicle for the programmer to specify actions to be executed by the machine, and it provides a set of concepts for the programmer to use when thinking about what can be done. The first purpose ideally requires a language that is “ close to the machine ” so that all important aspects of a machine are handled simply and efficiently in a way that is reasonably obvious to the programmer. The C language was primarily designed with this in mind. The second purpose ideally requires a language that is “ close to the problem to be solved ” so that the concepts of a solution can be expressed directly and concisely. The facilities added to C to create C++, such as function argument checking, const , classes, constructors and destructors, exceptions, and templates, were primarily designed with this in mind. Thus, C++ is based on the idea of providing both

- *direct mappings of built-in operations and types to hardware*to provide efficient memory use and efficient low-level operations, and

- *affordable and flexible abstraction mechanisms*to provide user-defined types with the same notational support, range of uses, and performance as built-in types.

This was initially achieved by applying ideas from Simula to C. Over the years, further application of these simple ideals resulted in a far more general, efficient, and flexible set of facilities. The result supports a synthesis of programming styles that can be simultaneously efficient and elegant.

The design of C++ has focused on programming techniques dealing with fundamental notions such as memory, mutability, abstraction, resource management, expression of algorithms, error handling, and modularity. Those are the most important concerns of a systems programmer and more generally of programmers of resource-constrained and high-performance systems.

By defining libraries of classes, class hierarchies, and templates, you can write C++ programs at a much higher level than the one presented in this book. For example, C++ is widely used in financial systems, for game development, and for scientific computation . For high-level applications programming to be effective and convenient, we need libraries. Using just the bare language features makes almost all programming quite painful. That's true for every general-purpose language. Conversely, given suitable libraries just about any programming task can be pleasant.

4.1.2 Language, Libraries, and Systems

The C++ fundamental (built-in) types, operators, and statements are those that computer hardware deals with directly: numbers, characters, and addresses. C++ has no built-in high-level data types and no high-level primitive operations. For example, the C++ language does not provide a matrix type with an inversion operator or a string type with a concatenation operator. If a user wants such a type, it can be defined in the language itself. In fact, defining a new general-purpose or application-specific type is the most fundamental programming activity in C++. A well-designed user-defined type differs from a built-in type only in the way it is defined, not in the way it is used. The C++ standard library provides many examples of such types and their uses. From a user's point of view, there is little difference between a built-in type and a type provided by the standard library. Except for a few unfortunate and unimportant historical accidents, the C++ standard library is written in C++. Writing the C++ standard library in C++ is a crucial test of the C++ type system and abstraction mechanisms: they must be (and are) sufficiently powerful (expressive) and efficient (affordable) for the most demanding systems programming tasks. This ensures that they can be used in large systems that typically consist of layer upon layer of abstraction.

Features that would incur run-time or memory overhead even when not used were avoided. For example, constructs that would make it necessary to store " " housekeeping information " in every object were rejected, so if a user declares a structure consisting of two 16-bit quantities, that structure will fit into a 32-bit register. Except for the new , delete , typeid , dynamic cast , and throw operators, and the try -block, individual C++ expressions and statements need no run-time support. This can be essential for embedded and high-performance applications. In particular, this implies that the C++ abstraction mechanisms are usable for embedded, high-performance, high-reliability, and real-time applications. So, programmers of such applications don't have to work with a low-level(error-prone, impoverished, and unproductive) set of language features.

C++ was designed to be used in a traditional compilation and run-time environment: the C programming environment on the UNIX system [UNIX,1985]. Fortunately, C++ was never restricted to UNIX; it simply used UNIX and C as a model for the relationships among language, libraries, compilers, linkers, execution environments, etc. That minimal model helped C++ to be successful on essentially every computing platform. There are, however, good reasons for using C++ in environments that provide significantly more run-time support. Facilities such as dynamic loading, incremental compilation, and a database of type definitions can be put to good use without affecting the language.

Not every piece of code can be well structured, hardware-independent, easy to read, etc. C++ possesses features that are intended for manipulating hardware facilities in a direct and efficient way without concerns for safety or ease of comprehension. It also possesses facilities for hiding such code behind elegant and safe interfaces.

Naturally, the use of C++ for larger programs leads to the use of C++ by groups of programmers. C++'s emphasis on modularity, strongly typed interfaces, and flexibility pays off here. However, as programs get larger, the problems associated with their development and maintenance shift from being language problems to being more global problems of tools and management.

C++ supports systems programming. This implies that C++ code is able to effectively interoperate with software written in other languages on a system. The idea of writing all software in a single language is a fantasy. From the beginning, C++ was designed to interoperate simply and efficiently with C, assembler, and Fortran. By that, I meant that a C++, C, assembler, or Fortran function could call functions in the other languages without extra overhead or conversion of data structures passed among them.

C++ was designed to operate within a single address space. The use of multiple processes and multiple address spaces relied on (extralinguistic) operating system support. In particular, I assumed that a C++ programmer would have the operating systems command language available for composing processes into a system. Initially, I relied on the UNIX Shell for that, but just about any " scripting language " will do. Thus, C++ provided no support for multiple address spaces and no support for multiple processes, but it was used for systems relying on those features from the earliest days. C++ was designed to be part of large, concurrent, multilanguage systems.

4.2 Random Number Generation Methods

4.2.1 Physical methods

The earliest methods for generating random numbers, such as dice, coin flipping and roulette wheels, are still used today, mainly in games and gambling as they tend to be too slow for most applications in statistics and cryptography.

4.2.2 Computational methods

Pseudo random number generators (PRNGs) are algorithms that can automatically create long runs of numbers with good random properties but eventually the sequence repeats. The series of values generated by such algorithms is generally determined by a fixed number called a seed. One of the most common PRNG is the linear congruential generator, which uses the recurrence $X_{n+1} = (aX_n + b)$ mod m. To generate numbers, where a, b and m are large integers, and is the next in X as a series of pseudo-random numbers. The maximum number of numbers the formula can produce is the modulus, m. To avoid certain non-random properties of a single linear congruential generator, several such random number generators with slightly different values of the multiplier coefficient a can be used in parallel, with a "master" random number generator that selects from among the several different generators. The method uses the following algorithm:

1. $X_{n+1} = (aX_n + b)$ mod c [given a seed value of X_o and integers a, b, c].

2. Let $X_n = X_{n+1}$ and repeat as necessary.

4.2.3 Generation from a probability distribution

here are a couple of methods to generate a random number based on a probability density function. These methods involve transforming a uniform random number in some way. Because of this, these methods work equally well in generating both pseudo-random and true random numbers. One method, called the inversion method, involves integrating up to an area greater than or equal to the random number (which should be generated between 0 and 1 for proper distributions). A second method, called the acceptance-rejection method, involves choosing an x and y value and testing whether the function of x is greater than the y value. If it is, the x value is accepted. Otherwise, the x value is rejected and the algorithm tries again.

4.2.4 Acceptance-rejection method (Von Neumann)

We suppose that for any given value of x, the probability density function f(x) can be computed, and further that enough is known about f(x) that we can enclose it entirely inside a shape which is C times an easily generated distribution h(x), as illustrated in Figure 4.1, Fig(a) illustration of the acceptance-rejection method. Random points are chosen inside the upper bounding figure, and rejected if the ordinate exceeds $f(x)$. Fig(b) illustrates a method to increase the efficiency.To generate f(x), first generate a candidate x according to $h(x)$. Calculate $f(x)$ and the height of the envelope $Ch(x)$; generate u and test if $uCh(x) \leqslant f(x)$. If so, accept x; if not reject x and try again.

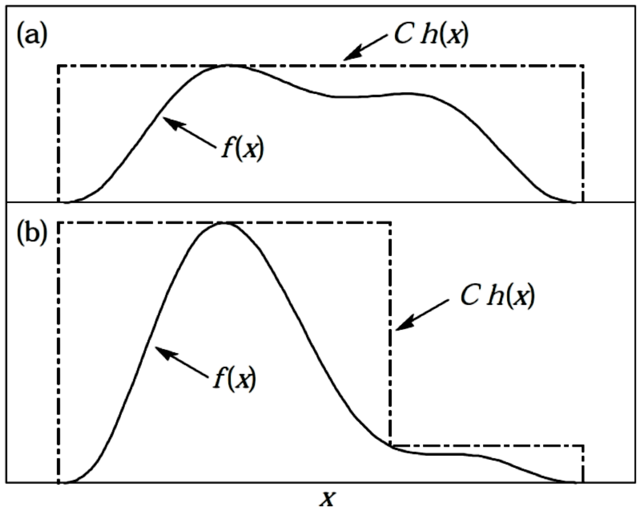

Figure 4.1: The acceptance-rejection methods

4.2.5 Mersenne Twister

One of the newest PRNG is the Mersenne Twister, originally developed by mathematicians Makoto Matsumoto and Takuji Nishimura in 1997. This PRNG is now often used in place of linear congruent generators like the example above. The generator runs faster than all but the least statistically sound pseudo random number generators. It appears to be distributed uniformly in 623 dimensions (!!) and has passed numerous tests for random. The Mersenne Twister gets its name from its huge period of 219937-1. This number is one of the largest known Mersenne primes. It is believed that the Mersenne Twister would probably take longer to cycle than the entire future existence of humanity (and, perhaps, the universe). However, if you observe enough numbers generated by the Mersenne Twister, the generator would allows all future numbers to be predicted; therefore, the Mersenne Twister is not suitable in cryptography. This illustrates the fact that no single PRNG is the best choice for all applications. The commonly-used version of Mersenne Twister, MT19937, which produces a sequence of 32-bit integers, has the following desirable properties:

1. It has a very long period of 219937 - 1. While a long period is not a guarantee of quality in a random number generator, short periods (such as the 232 common in many older software packages) can be problematic.

2. It is reliable than others.

3. It is fast and economical.

4. This has a huge k-distribution property to v-bit accuracy for each v, v =1, 2. . . 32. It is a measure of the randomness of the numbers produced.

5. This generator is developed for generating [0, 1]-uniform real random numbers, with special attention paid to the most significant bits.

4.3 Random number generation by computers

It may be wondering how a computer can actually generate a random number. Where does this " randomness " come from? It's just a piece of computer code. We generally group the random numbers computers generate into two types, depending on how they're generated: " True " random numbers and pseudo-random numbers.

To generate a " true " random number, the computer measures some type of physical phenomenon that takes place outside of the computer. For example, the computer could measure the radioactive decay of an atom. According to quantum theory, there's no way to know for sure when radioactive decay will occur, so this is essentially " pure randomness " from the universe. An attacker wouldn't be able to predict when radioactive decay would occur, so they wouldn't know the random value.

Pseudo random numbers are an alternative to " true " random numbers. A computer could use a seed value and an algorithm to generate numbers that appear to be random, but that are in fact predictable. The computer doesn't gather any random data from the environment.This isn't necessarily a bad thing in every situation. For example, if you're playing a video game, it doesn't really matter whether the events that occur in that game are cased by " true " random numbers or pseudo random numbers. The production of random numbers has been an important application for computers since the beginning of the modern era of computation. The random number library provides classes that generate random and pseudo-random numbers. In header <cstdlib> (the C++ version of C's <stdlib.h>) we find:

- RAND MAX, a macro that produces integer random number in the range of 0 and a max.

•std::rand (), a function that produces a pseudo-random number in the closed interval [0; RAND MAX]; and

•std::srand (), a function to initialize (seed) a new sequence of such numbers.

But, they suffer poor quality and performance. To address the above shortcomings, the C++11 standard provides a new header; ¡random¿.The traditional term random number generator encompasses two kinds of functionality. However, C++ ¡random¿ draws a clear distinction between the two: The random number library provides classes that generate random and pseudo-random numbers. These classes include:

•Random number engines: (both pseudo-random number generators, which generate integer sequences with a uniform distribution, and true random number generators if available)

•Random number distributions: (e.g. uniform, normal, or poison distributions) which convert the output of random number engines into various statistical distributions.

An engine's role is to return unpredictable (random) bits, ensuring that the likelihood of next obtaining a 0 bit is always the same as the likelihood of next obtaining a 1 bit. A distribution's role is to return random numbers (variates) whose likelihoods correspond to a specific shape. E.g., a normal distribution produces variates according to a " bell-shaped curve." A user needs an object of each kind in order to obtain a random variate. The purpose of the engine is to serve as a source of randomness.

4.3.1 Engines in the standard library

The C++11 standard library provides, in **default_random_engine**, engine types aimed at several different kinds of users.

•The library provides default random engine, an alias for an engine type selected by the library vendor, according to the standard, " on the basis of performance, size, quality, or any combination of such factors, for relatively casual, inexpert, and/or lightweight use ".

The library provides pre-configured engine types of known good quality. Some of which are:

•Linear congruential engines: **minstd_rand0, minstd_rand**

•Mersenne twister engines: **mt19937, mt19937_64** etc.

4.3.2 Random number distributions

A random number distribution post-processes the output of an random number engine in such a way that resulting output is distributed according to a defined statistical probability density function. Defined in header $< random >$

Uniform distributions

uniform_int_distribution	produces integer values evenly distributed across a range
uniform_real_distribution	produces real values evenly distributed across a range

Normal distributions

normal_distribution	produces real values on a standard normal (Gaussian) distribution.
lognormal_distribution	produces real values on a lognormal distribution.
chi_squared_distribution	produces real values on a chi-squared distribution.

4.4 Multi-threading

Multi-threading [4] is a specialized form of multitasking and a multitasking is the feature that allows your computer to run two or more programs concurrently. In general, there are two types of multitasking: process-based and thread-based. Process-based multitasking handles the concurrent execution of programs. Thread-based multitasking deals with the concurrent execution of pieces of the same program. A multi-threaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. C++ does not contain any built-in support for multi-threaded applications. Instead, it relies entirely upon the operating system to provide this feature.

In multi-threading environment processor fork the main program into different threads and execute concurrently, then system goes to a waiting stage until every thread complete executing corresponding tasks, after that it join all threads to the main program.Multi-threading logic is show in the figure 4.2.Important point to be noted is that every thread shares same memory.

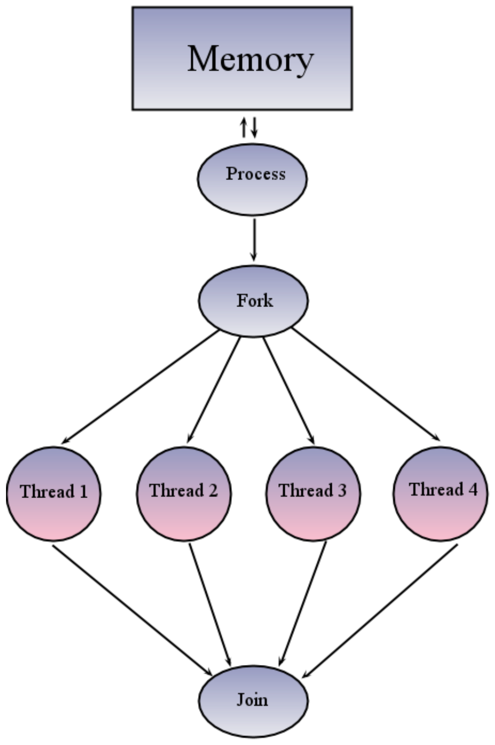

Figure 4.2: Multi-thread process

Recent processors have 4 CPUs or core, Each thread is executed in different core. This concurrent execution allows high performance computation so that execution time can be saved and degree of freedom will be increased

Here we are implementing this freedom in our algorithm. Two chains are generated concurrently in two different core and employed third core to compute R-factor of Gelman-Rubin diagnostics. Main function is forked into threads and terminated with necessary condition.Block diagram of threading is shown in the figure 4.3

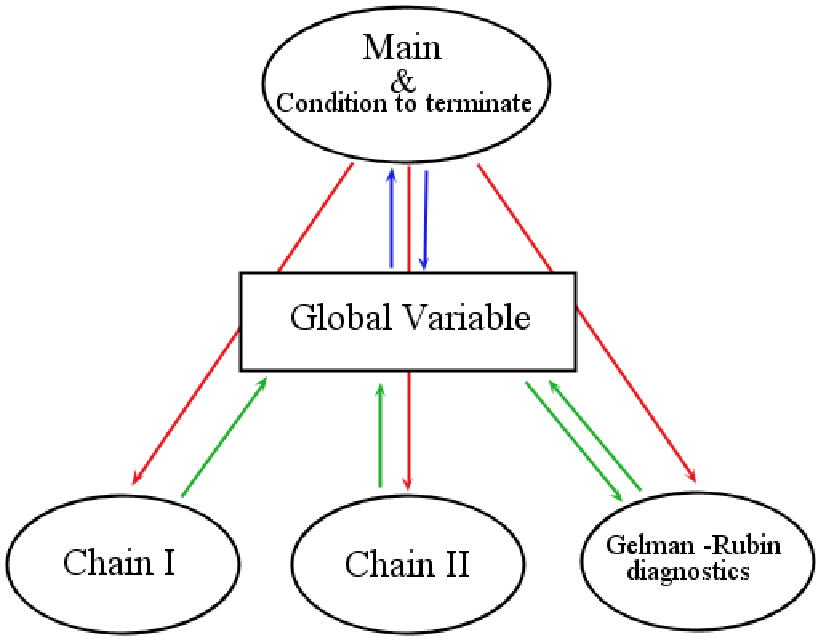

Figure 4.3: Block Diagram of MCMC chain generation

4.5 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Here python is used for data analysis, the used modules are:

• *numpy* - It is the module package for scientific computing

• *matplotlib.pyplot* - It is a collection of command style functions that make matplotlib work like MATLAB inside python.

Chapter 5

Data Analysis Tests

5.1 Why Testing is needed?

Before using the algorithm to address some physical problems, its required to test algorithm with fake data of known parameter. Here we generate fake data of $y(x)$ with error dy, then runs program with this data and check how precisely the program converges to the parameter we expected.Unless until algorithm gives the expected parameter value, we cannot trust algorithm.So here we are using four functions

1.*Linear function*

2.*Parabolic function*

3.*Quintic function*

4.*Rosenbrock function*

5.2 Fitting Linear Equation

A linear equation is an algebraic equation in which each term is either a constant or the product of a constant and a single variable.Linear equation in the two variables x and y is of the form

$$y = mx + c$$

where m is the slope of the the function, it is defined as

$$m = \frac{dy}{dx},$$

and c is the intercept of the function with x axis. A graphed function is shown in Fig 5.1

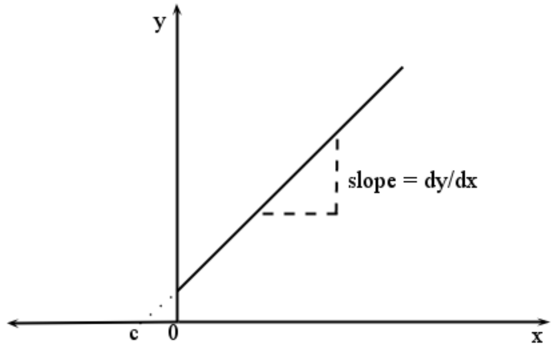

Figure 5.1: Linear Equation

Data for linear equation with $m = 0.7$ and $c = -0.1$ is given in the table 5.1

Table 5.1: Data of a Linear function

x	y	dy
0	0.107	0.160
1	0.524	0.192
2	1.391	0.170
3	2.028	0.137
4	2.877	0.154
5	3.332	0.153
6	4.143	0.049
7	5.084	0.172
8	5.733	0.018
9	6.410	0.130

Table 5.2: R values calculated while fitting linear function

R_a	R_b
1.3047	1.36417
1.05914	1.2735
1.00837	1.12062
0.999527	1.06028
0.999002	1.01988

Code will run with data given in Table 5.1 and will terminate for a particular R value, usually R value for termination is set to 1.03, R values printed on running terminal is shown in Table 5.2. Program started to run with initial values, then R value for first 100 generated parameter is estimated,*i.e* $R_a = 1.3047$ and $R_b = 1.36417$. Termination was called for the values $R_a = 0.999002$ and $R_b = 1.01988$.

Where R_a and R_b are the corresponding R values of m and c respectively. Program terminates only when each R values corresponding to parameters reaches desired value. Obtained χ^2, m and c are:

χ^2	m	c
5.2721e+08	49.999	11.613
5.2704e+08	49.992	11.604
5.2683e+08	49.984	11.584
5.2664e+08	49.975	11.582
⋮	⋮	⋮
5.5589e+06	5.1455	6.0374
5.5559e+06	5.1431	6.0451
⋮	⋮	⋮
10.585	0.73726	-0.17282
10.896	0.73887	-0.17899 0
10.562	0.73743	-0.17596 0
10.534	0.73473	-0.15736

Table 5.3: Parameters of linear function generated by MCMC algorithm

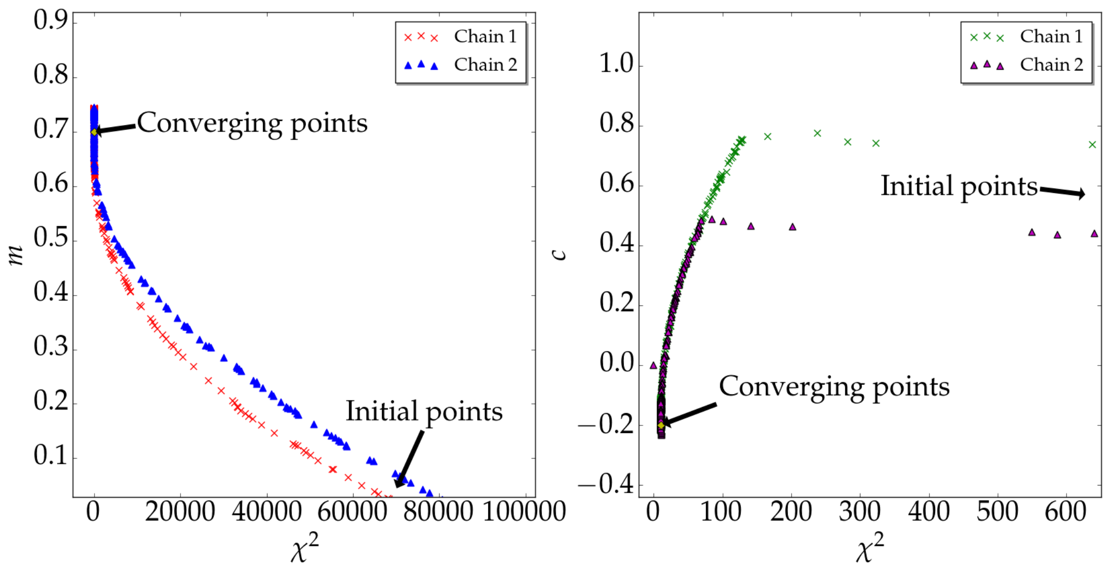

Figure 5.2: Markov chains, generated in parameter space

Simply, we are minimizing the χ^2 to find the best fit parameters, but as you can see in above array $\chi^2 \approx 10$ and χ^2 cannot be reduced further. We have 10 data points and this χ^2 is the sum of all this ten points, due to this reason minimizing χ^2 further is difficult. It is also depend on error of function, if error is negligible, then χ^2 can be reduced further.

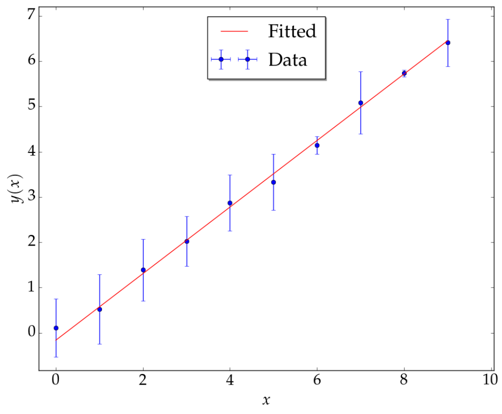

Figure 5.3: Data and Fitted plot of linear function

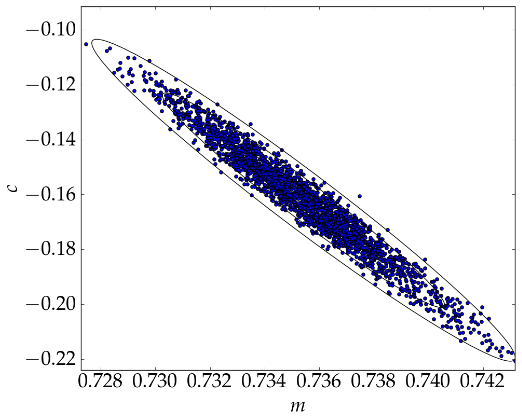

Figure 5.4: Confidence region of m and c with $1\sigma, 2\sigma$ and 3σ contours

5.3 Fitting Parabolic Equation

The parabola is the locus of points in that plane that are equidistant from both the directrix and the focus. Another description of a parabola is as a conic section, created from the intersection of a right circular conical surface and a plane which is parallel to another plane which is tangential to the conical surface.Algebraically defined by equation

$$y = ax^2 + bx + c$$

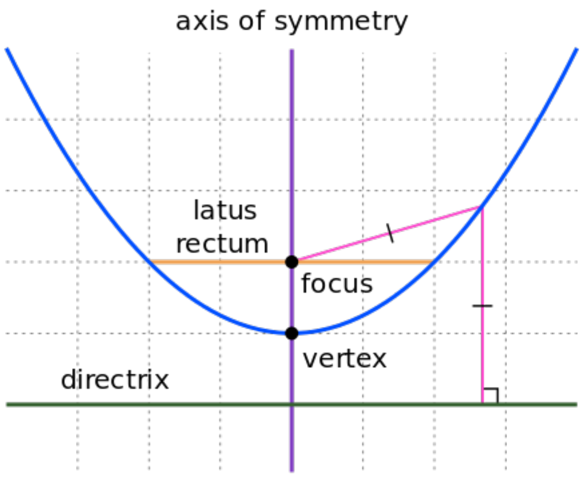

Figure 5.5: Parabola

Data with parameters value $a = 1, b = 0$ and $c = 0$ is given in table 5.4.

Procedures for parameter estimation is same as done before for linear fitting.Here we want to generate three parameter, hence three R values is needed to be minimized to desired value. Again R value is set to 1.03, so that program will terminate when R runs less than 1.03 *i.e*

$$R_a \ \& \ R_b \ \& \ R_c < 1.03$$

Table 5.4: Data of a Parabolic function

x	y	dy
-5	25	0.285482
-4	16	0.202485
-3	9	0.143875
-2	4	0.110454
-1	1	0.0974679
0	0	0.0948683
1	1	0.0974679
2	4	0.110454
3	9	0.143875
4	16	0.202485

Table 5.5: R values calculated while fitting Parabolic function

R_a	R_b	R_c
1483.9	2197.24	1745.65
864.451	1256.54	1160.88
334.852	424.459	647.607
7.66922	13.1516	12.556
5.40155	10.0435	7.5939
4.60538	8.87144	5.88995
1.16719	1.95582	1.0024
1.00964	1.03834	1.00672
1.00789	1.03182	1.00645
1.0064	1.02633	1.00608

From Table 5.5: There are three rows corresponding to R_a, R_b and R_c. Initial values of R are very high but as program runs R value decreases and termination was called for $R_a = 1.0064, R_b = 1.02633$ and $R_c = 1.00608$

Generated array of parameters with respective χ^2 is shown in Table 5.6.Program started execution with $a = -48.595, b = -81.83, c = 20.859$. For this initial parameters, computed χ^2 equal to 76902000. In the next iteration χ^2 is reduced to 76881000, and it keeps on decreasing χ^2. As a parallel process, potential reduction factor R, will be reduced accordingly.

χ^2	a	b	c
76902000	-48.595	-81.836	20.859
76881000	-48.585	-81.839	20.838
76864000	-48.581	-81.811	20.828
⋮	⋮	⋮	⋮
2389.6	0.72405	-0.20321	2.446
2326.8	0.71957	-0.21757	2.392
⋮	⋮	⋮	⋮
10.5511	1.0069	0.01875	-0.030794
10.9007	0.99974	0.015004	-0.017332
10.7651	0.99456	0.0010385	0.0065017

Table 5.6: Generated parameters of parabolic function

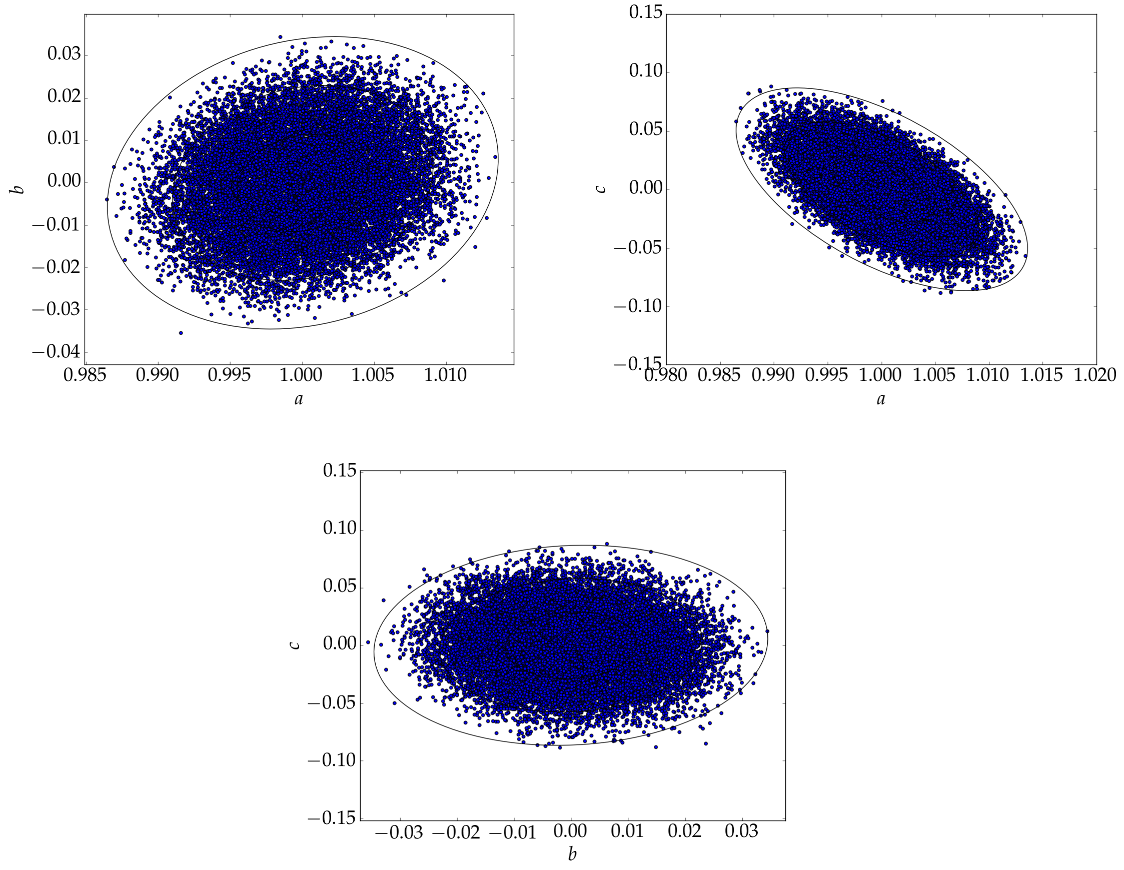

Figure 5.6: Parameter distributions of a, b and c in 1σ,2σ and 3σ contours

5.4 Quintic function Fitting

In algebra, a quintic function[6] is a function of the form

$$y(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$$

where a, b, c, d, e and f are members of a field, typically the rational numbers, the real numbers or the complex numbers, and a is nonzero. In other words, a quintic function is defined by a polynomial of degree five. If a is zero but one of the coefficients b, c, d, or e is non-zero, the function is classified as either a quartic function, cubic function, quadratic function or linear function.Because they have an odd degree, normal quintic functions appear similar to normal cubic functions when graphed, except they may possess an additional local maximum and local minimum each. The derivative of a quintic function is a quadratic function.

Table 5.7: Data of a Quintic function

x	y	dy
-3	-51	4.534
-2.3	1.51	1.487
-1.6	1.35	1.141
-1	3	0.264
-0.33	2.95	0.247
-0.33	9.04	0.760
1	9	0.731
1.6	-1.51	0.131
2.33	-3.13	0.211
3	63	4.403

Table 5.8: R values calculated while fitting Quintic function

R_a	R_b	R_c	R_d	R_e
11.2372	13.1448	94.3199	10.7191	315.749
1.5112	0.997052	2.77041	1.08407	171.487
0.995106	1.02321	1.82977	1.06203	25.5059
1.03568	1.03727	1.27715	1.08422	1.11694
1.01043	1.01099	1.0176	1.01411	1.01091
1.00652	1.00713	1.00162	1.00814	1.00313
1.00328	1.00359	0.9993	1.00366	1.00046
1.00014	1.00017	0.999442	1.00019	0.999645
0.999914	0.9999	0.999264	0.999887	0.999761

Data with parameter values $a = 1, b = -8, c = -0.1, d = 10$, and $e = 6$ is given in Table 5.5. Parameter f is set to zero , hence f is neglected in estimation.Termination of chain is called when R values run less than unity $i.e$

$$R_a \ \& \ R_b \ \& \ R_c \ \& \ R_d \ \& \ R_e < 1$$

So termination was called for the last R values given in the Table 5.8. Table 5.9 is the chain of parameters.

χ^2	a	b	c	d	e
182832	0.988965	0.687632	0.325444	0.256911	0.111319
176543	0.965941	0.693969	0.332459	0.254442	0.119431
143455	0.852972	0.676924	0.343194	0.217744	0.062651
⋮	⋮	⋮	⋮	⋮	⋮
10.10023	0.95819	-7.62471	-0.118096	9.48343	5.98834
10.25506	0.955749	-7.61645	-0.122666	9.49639	5.99731
10.21182	0.956938	-7.62684	-0.125125	9.51851	5.99369

Table 5.9: Parameters chain of Quintic function

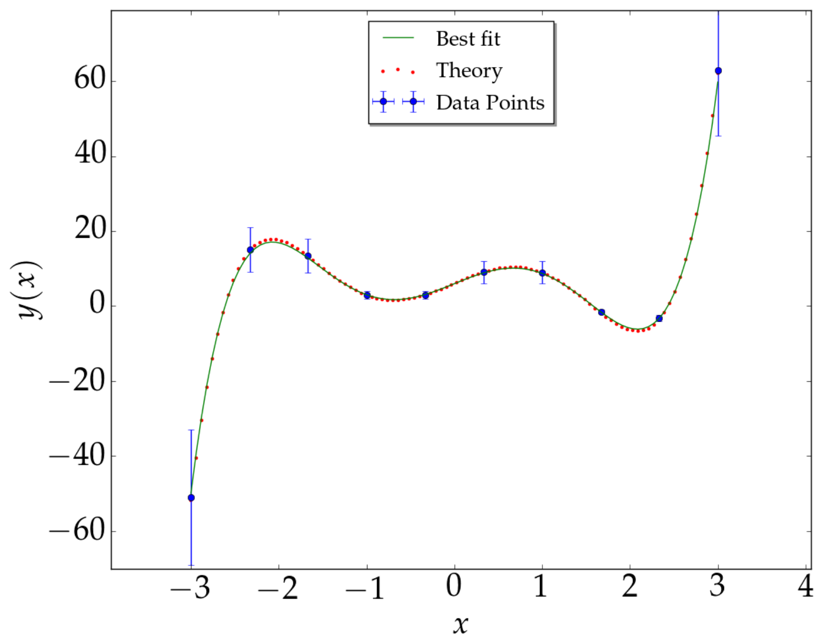

Figure 5.7: Data and Fitted curve of Quintic function

It is clear from Figure 5.8 that parameters estimated with data points in Table 5.5 is very consistent with theoretical values.Confidence region of all parameters are shown Figure 5.9.Estimated parameters are:

$$a = 0.956 \pm 0.018 \qquad b = -7.626 \pm 0.04, \qquad c = -0.125 \pm 0.011$$

$$d = 9.518 \pm 0.025 \qquad e = 5.993 \pm .01.$$

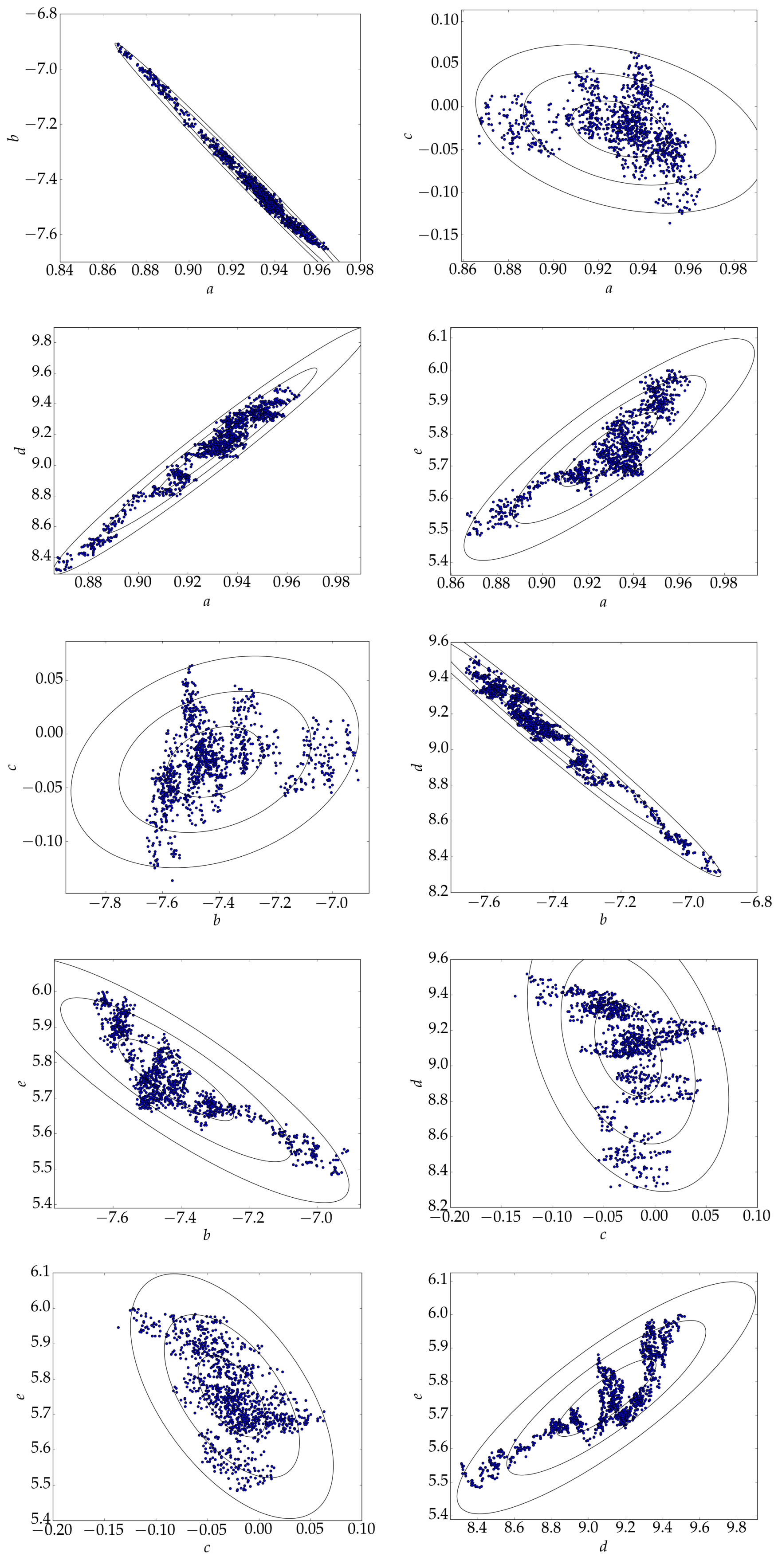

Figure 5.8: Confidence region plots of Quintic function parameters

5.5 Fitting Rosenbrock Funtion

In mathematical optimization, the Rosenbrock function[6] is a non-convex function used as a performance test problem for optimization algorithms introduced by Howard H. Rosenbrock in 1960.It is also known as Rosenbrock's valley.The global minimum is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial. To converge to the global minimum, however, is difficult. The function is defined by

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

It has a global minimum at$(x, y) = (a, a^2)$, where $f(x, y) = 0$. Usually $a = 1$ and $b = 100$.

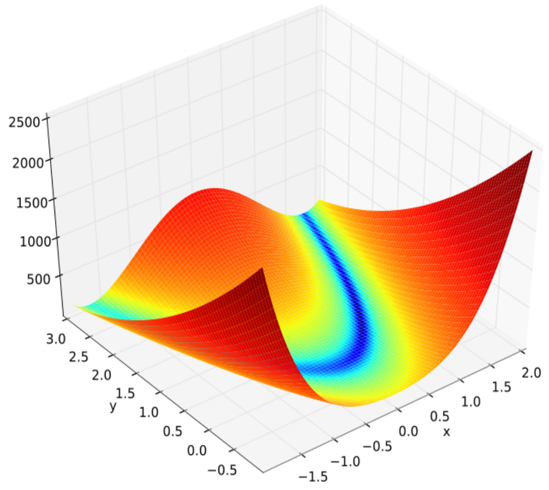

Figure 5.9: 3D Rosenbrock Funtion. Image credit: Wikipedia

Data of Rosenbrock's function is given in Table 5.10. Important point to be noted here is that, for every other functions we minimized χ^2,whereas here function $f(x)$ itself is minimized.

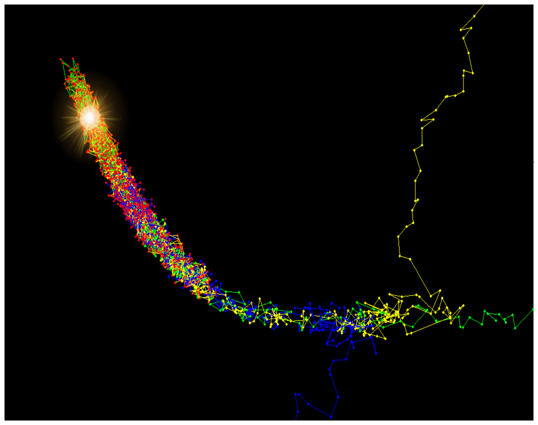

Figure 5.10: Rosenbrock's function running on MH algorithm. Image credit: Wikipedia

Table 5.10: Data of a Rosenbrock's function

x	y	df
1.02886	-0.987112	41.84
0.292123	-0.506892	3.55
-0.38354	-1.73882	35.75
-0.105525	-1.34925	18.62
1.37729	-0.693344	67.10
-1.34056	0.535644	16.45
0.293989	0.814933	5.35
0.60927	1.23561	7.53
0.391069	1.45177	16.90
0.4606	0.836524	3.92

Table 5.11: Minmized values of function with corresponding parameters

$function$	a	b
1211159.875	49.999	2.068
1204279.875	49.992	1.974
1187709.875	49.975	1.754
⋮	⋮	⋮
0.7589591742	2.185754538	96.611
0.7075862288	2.157474518	96.735
⋮	⋮	⋮
0.2956601679	1.283427477	101.141
0.2104740739	1.194565654	101.086
0.1906311065	1.00248754	100.989

Function values with corresponding parameter values is shown in Table 5.11, The function is minimized to a value of 0.190 and thus reached the desired values of a and b. A 3D plot of generated chain with z-axis as function value, x-axis as a and y-axis as b is shown in Fig 5.11

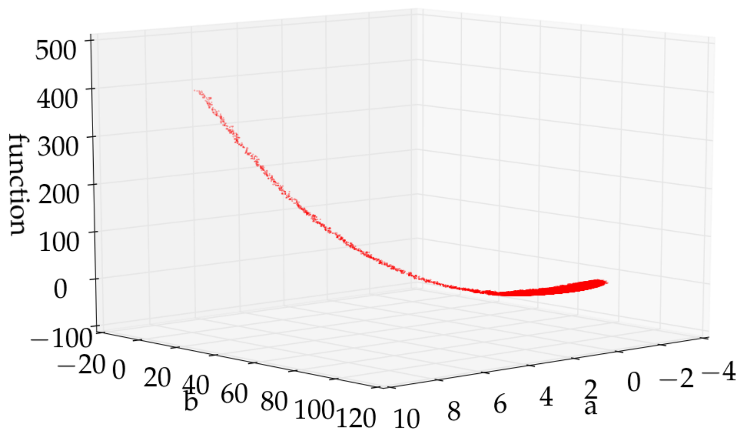

Figure 5.11: 3D Plot of *parameter space* vs. *function value* of Rosenbrock's function

Chapter 6

Cosmological Models

6.1 General Relativity

Mass m that appears in the second law of Newtonian mechanics is the inertial mass. The same mass appears in the right hand side of the Newton's law of gravity, where it determines the strength of gravitational force. There is no mathematical reason for this. In fact, by analogy with electromagnetism, the law of gravity could involve not the inertial mass but a different quantity, that could be called gravitational charge or gravitational mass, m_g. Some particles could have it, some not. Two particles with the same inertial mass could have different gravitational mass. However, in Nature

$$m_g = m \tag{6.1}$$

Thus, Eq 6.1 is a physical law which is incorporated in the Newton's law of gravity. It is called the Principle of Equivalence of inertial and gravitational mass. Because of this law "particles" of very different masses have exactly the same acceleration in a given gravitational field and provided they have the same initial velocity they exhibit identical motion. This suggests that particle motion in gravitational field has nothing to do with the properties of particles but is rather determined by the properties of the spacetime itself. In fact, Albert Einstein concluded that gravity makes itself felt via warping the spacetime. Indeed, the Minkowskian spacetime of Special Relativity is flat. By definition, a metric space is called flat if there exist such global coordinates, x^v , that its metric form is the sum of $\pm(dx^v)^2$ in every point of the space. Otherwise, it is called curved or warped. For example, a Euclidean plane is a flat two dimensional space. Indeed, in Cartesian coordinates its metric form is

$$dl^2 = dx^2 + dy^2$$

everywhere. A sphere in Euclidean space is not flat (in fact, it is not even a space but a manifold). It is impossible to introduce such coordinates on the sphere such that the distance between any two of its infinitesimally close points is given by the above metric form.

So, in most problems the metric form is not known before hand. So often it is written in the most general form

$$ds^2 = \sum_{v=0}^{3} \sum_{\mu=0}^{3} g_{v\mu} dx^v dx^\mu \tag{6.2}$$

Here x^v are some coordinates of spacetime (traditionally the indexes vary not from 1 to 4 but from 0 to 3), and the coefficients $g_{v\mu}$ are functions of these coordinates. In fact, these coefficients are the components of the so-called metric tensor.

The key equation of General Relativity is the celebrated Einstein's equation

$$R_{v\mu} = \frac{1}{2} R g_{v\mu} + \frac{8\pi G}{4} T_{v\mu} \tag{6.3}$$

which Einstein discovered in 1915. Here $R_{v\mu}$ is the Ricci tensor, R is the scalar curvature they describe properties of spacetime. In Minkowskian spacetime $R_{v\mu} = 0$ and $R = 0$ everywhere. $T_{v\mu}$ is called the stress- energy-momentum tensor. It describes the distribution of energy, momentum, and stresses associated with matter, radiation, and all sorts of force fields. The simple appearance of Einstein equation is deceptive. In fact, this is one of the most difficult equations of Mathematical Physics, which can be solved analytically only in very limited cases of highly symmetric problems. Only during the last decade mathematicians figured out how to solve this equation numerically. So here I only briefly discuss its nature.

Since $v, \mu = 0...3$ we have sixteen equations in Eq.6.3 The Ricci tensor and the curvature scalar are functions of $g_{v\mu}$, its first and second order derivatives with respect to all four coordinates. Thus, we are dealing not with one equation but with a system of sixteen simultaneous second order partial differential equations. The unknown functions of coordinates in these equations are $g_{v\mu}$ and $T_{v\mu}$ – the Einstein equation not only describes how matter warps spacetime but also how matter evolves in this warped spacetime. If correct, the Einstein equation should also describe the Universe, which is filled with gravitationally interacting matter. In 1915 the Universe and the Milky Way (our Galaxy) were considered as the same thing, and the Milky Way appeared to be very much static. Einstein analyses models of static Universe and concluded that such Universe cannot be infinite. Instead, it must be finite but without boundaries - the space must be wrapped onto itself like a sphere in Euclidean geometry. Later however, he run into difficulty as his equation 6.3 did not allow such a static solution. So he modified this equation by adding the so called Cosmological term:

$$R_{v\mu} = \frac{1}{2}Rg_{v\mu} + \frac{8\pi G}{4}T_{v\mu} - \frac{\Lambda}{c^2}g_{v\mu} \tag{6.4}$$

where Λ is called the Cosmological constant. (In Newtonian Physics this is equivalent to introduction of a repulsive force to balance gravity.) After the Hubble's discovery Einstein abandoned his work on the Cosmological term, and called it his " biggest blunder ". However, the modern data indicate that the Cosmological term is in fact needed to explain the observations.

6.2 Friedmann's equations

The simplest from the mathematical view point model of the Universe is where it is uniform, that is where its geometry is the same at every point. From the observational prospective it is not obvious that our Universe is uniform. Indeed, the very existence of the boundary of visible Universe (the CMB "screen") seem to indicate that it is not uniform. However, as we shell see later this observations are easily explained in the relativistic models of the Universe where it is assumed uniform. These models predict the observed Universe to appear isotropic (or almost isotropic) and this prediction agrees with the observations very well indeed.

The spacetime metric form of such Universe can be written as

$$ds^2 = -c^2dt^2 + R^2\left[\frac{d\chi^2}{1-k\chi^2} + \chi^2(d\theta^2 + sin^2\theta d\phi^2)\right] \tag{6.5}$$

Metric Eq.6.5 is known as the *Robertson-Walker metric.*

In the derivation we assume that the distribution of both matter and radiation can be described as continuous (on scales above the size of voids). That is both are treated as a uniform gas (compressible fluid) with the mass-energy density ρ and pressure P.

Out of 16 coefficients $g_{v\mu}$ only four are non-vanishing in the Robertson-Walker metric: $g_{tt} = -c^2, g_{\chi\chi} = R^2(t)/(1-k\chi^2), g_{\theta\theta} = R^2(t)\chi^2$ and $g_{\phi\phi} = R^2(t)\chi^2 sin^2\theta$ and they include only one unknown function, the scaling factor $R(t)$. After substituting these expressions into the Einstein's equations (6.3) with the stress-energy-momentum tensor of ideal fluid one finds that out of these 16 equations only two are independent. They are

$$\left(\frac{\dot{R}}{R}\right)^2 + \frac{kc^2}{R^2} = \frac{8\pi}{3}G\rho \tag{6.6}$$

and

$$2\frac{\ddot{R}}{R} + \left(\frac{\dot{R}}{R}\right)^2 + \frac{kc^2}{R^2} = -\frac{P}{c^2}8\pi G, \tag{6.7}$$

where ρ is the mass-energy density and P is the pressure. Equation 6.6 is known as the *Friedmann equation*, after the Russian mathematician Alexander Friedmann who first derived it back in 1922. Subtracting Eq.6.6 from Eq.6.7 one obtains the so-called *acceleration equation*

$$\frac{\ddot{R}}{R} = -\frac{4\pi G}{3}\left(\rho + 3\frac{P}{c^2}\right) \tag{6.8}$$

When the Einstein equation with the cosmological term (Eq.6.4) is used instead of the original Eq.6.4, the Friedmann and the acceleration equations become

$$\left(\frac{\dot{R}}{R}\right)^2 + \frac{kc^2}{R^2} = \frac{8\pi}{3}G\rho + \frac{\Lambda}{3} \tag{6.9}$$

$$\frac{\ddot{R}}{R} = -\frac{4\pi G}{3}\left(\rho + 3\frac{P}{c^2}\right) + \frac{\Lambda}{3} \tag{6.10}$$

6.3 Critical density

Since the stationary Universe is physically impossible, and seems to be in conflict with the observations of distant galaxies, it makes sense to go back to the cosmological equations without the cosmological constant and explore their solutions. This is exactly where the attention of cosmologists turned to after the Hubble's discovery.

Consider the Friedmann equation and the acceleration equation without the cosmological constant. Since both for matter and radiation $\rho > 0$ and $P > 0$, the acceleration equation shows that $\ddot{R} < 0 (q_0 > 0)$. Hence, we conclude that, without the cosmological constant, Universe's expansion must be slowing down!.

What is even more interesting, the Friedmann equation shows that the actual geometry of the Universe without the cosmological constant can be deduced from it's current density ρ_0 and the Hubble constant. Indeed, from Eq.6.6 we have that

$$H_0^2 = \frac{8\pi}{3}G\rho_0 - \frac{kc^2}{R_0^2}, \tag{6.11}$$

or

$$\rho_0 - \rho_c = \frac{3c^2}{8\pi G R_0^2}k, \tag{6.12}$$

where

$$\rho_c = \frac{3H_0^2}{8\pi G}. \tag{6.13}$$

Now one can see that the Universe is closed if $\rho_0 > \rho_c$, flat if $\rho_0 = \rho_c$, and open if $\rho_0 < \rho_c$. For this reason ρ_c is called the critical density. It is convenient to describe how close the current density in the Universe to the critical value by the dimensionless parameter

$$\Omega_0 = \frac{\rho_0}{\rho_c} \tag{6.14}$$

It is called the critical parameter. If $\Omega_0 < 1$ then $\rho_0 < \rho_c$ and $k < 0$ the Universe is open. If $\Omega_0 > 1$ then $\rho_0 > \rho_c$ and $k > 0$ the Universe is closed. If $\Omega_0 = 1$ then the Universe is flat.

6.4 The Friedmann models

Will the deceleration be able eventually to stop the expansion and turn it into a contraction? At this point $\dot{R}$ would vanish. However, Eq.6.11 shows that this is not possible if k $\leq$ 0. Thus, both the flat and the open Universes are destined to expand forever. For the closed Universe this equation suggests that it may be possible to reach the state where $\dot{R} = 0$. Since $\ddot{R} < 0$ this is a maximum of $R(t)$ and the Universe will begin to contract after this point. Since at present the radiation makes only a very small contribution to the energy-density, it makes sense to ignore it and consider the models with $P = 0$. This is exactly what was done by Alexander Friedmann in his study. In this case the Friedmann equations read

$$\left(\frac{\dot{R}}{R}\right)^2 = \frac{8\pi}{3}G\rho - \frac{kc^2}{R^2}, \tag{6.15}$$

$$\frac{\ddot{R}}{R} = -\frac{4\pi G}{3}\rho, \tag{6.16}$$

and

$$\dot{\rho} = -3\frac{\dot{R}}{R}\rho. \tag{6.17}$$

Integrating Eq.6.17 we find that

$$\rho R^3 \rho_0 R_0^3 = constant. \tag{6.18}$$

Substituting this result in Eg.6.15 we obtain differential equation for R

$$\dot{R}^2 = c^2\left(\frac{\alpha^2}{R} - k\right), \tag{6.19}$$

where

$$\alpha^2 = \frac{8\pi G}{3}\frac{\rho_0 R_0^3}{c^2}. \tag{6.20}$$

Before this we note that from Eqs.6.16 it follows that

$$\frac{\ddot{R}_0}{R_0} = -\frac{4\pi G}{3}\rho_0, \tag{6.21}$$

When combined with the definition of the deceleration parameter, this result means that

$$\rho_0 = \frac{3H_0^2}{4\pi G}q_0 = 2q_0\rho_c. \tag{6.22}$$

Thus, in all Friedmann's models

$$\Omega_0 = 2q_0 \tag{6.23}$$

6.4.1 Flat Universe

In this case $k = 0, \Omega_0 = 1, q_0 = 1/2$, and Eq.6.19 reads

$$\dot{R}^2 = \frac{\alpha^2 c^2}{R}. \tag{6.24}$$

Integrating this equation we find

$$R^{3/2} = at + C, \tag{6.25}$$

where $a = \sqrt{\alpha^2 c^2}$ and C is the integration constant. From this it is clear that at some time t_{BB} we have $R = 0$. Resetting the clocks so this time becomes $t = 0$ (which is equivalent to imposing the boundary condition $R(0) = 0$) we obtain

$$R = At^{2/3}, \tag{6.26}$$

where A is constant. Thus, the Universe expands forever.

6.5 Basic Equation of the Dark energy models

The Friedmann equation of the ΛCDM [3] model with spatial curvature can be written as

$$H^2(z, H_0, p) = H_0^2\left[\Omega_{m0}(1+z)^3 + \Omega_\Lambda + (1 - \Omega_{m0} - \Omega_\Lambda)(1+z)^2\right], \tag{6.27}$$

where z is the redshift, $H(z, H_0, p)$ is the Hubble parameter, H_0 is the Hubble constant, and the model-parameter set is $p = (\Omega_{m0}, \Omega_\Lambda)$ where Ω_{m0} is the nonrelativistic (baryonic and cold dark) matter density parameter and Ω_Λ that of the cosmological constant.And $\Omega_\Lambda + \Omega_{m0} = \Omega_k$. Throughout, the subscript 0 denotes the value of a quantity today. For $k = 0$, $\Omega_k = 1$ and the last term in the Eq 6.27. get vanished and hence,

$$H^2(z, H_0, p) = H_0^2\left[\Omega_{m0}(1+z)^3 + (1 - \Omega_{m0})\right] \tag{6.28}$$

Chapter 7

Result and Conclusion

In chapter 5, we generated data with putting parameter values, then using that data, program is executed. Chain generated in parameter space converged to values which are very consistent with known parameters. So it confirms that our program is ready for addressing real problems. Chapter 6 introduced concept of cosmological models.In this chapter we are going to fit Lambda Cold Dark-matter Model(ΛCDM)

7.1 Parameter estimation of Curved ΛCDM model with parameter set $p = (H_0, \Omega_{m0}, \Omega_\Lambda)$

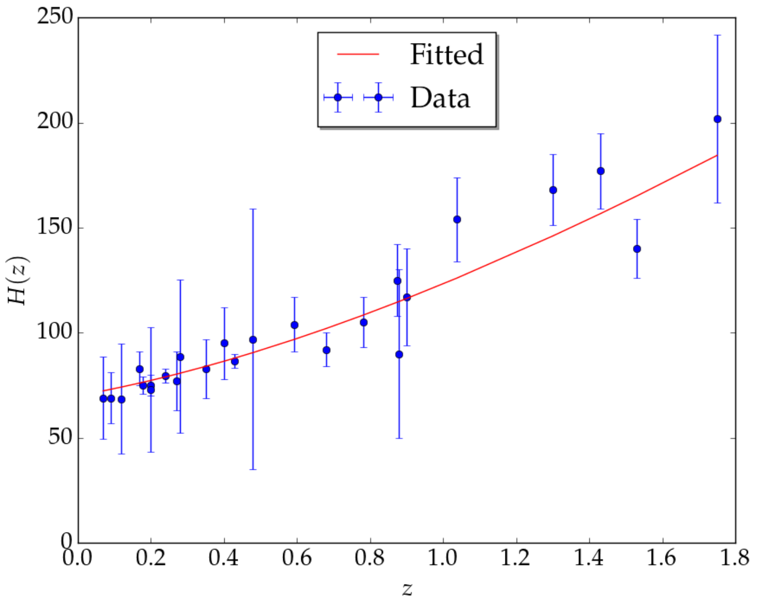

Figure 7.1: Data and fitted plot of OHD

Data were collected from different articles, all available OHD is tabulated in Table 7.1. In Figure 7.1 data points is plotted with corresponding errorbars. 2D Confidence region of parameters $\Omega_m, \Omega_\Lambda, \Omega_k$ and H_0 is shown in figure 7.2

z	$H(z)$	$\sigma_{H(z)}$	$Reference$
0.090	69	12	[7]
0.170	83	8	[16]
0.270	77	14	[16]
0.400	95	17	[16]
0.900	117	23	[16]
1.300	168	17	[16]
1.430	177	18	[16]
1.530	140	14	[16]
1.750	202	40	[16]
0.480	97	62	[13]
0.880	90	40	[13]
0.179	75	4	[9]
0.199	75	5	[9]
0.352	83	14	[9]
0.593	104	13	[9]
0.680	92	8	[9]
0.781	105	12	[9]
0.875	125	17	[9]
1.037	154	20	[9]
0.24	79.69	3.32	[12]
0.43	86.45	3.27	[12]
0.07	69.0	19.6	[17]
0.12	68.6	26.2	[17]
0.20	72.9	29.6	[17]
0.28	88.8	36.6	[17]

Table 7.1: Observational Hubble Data

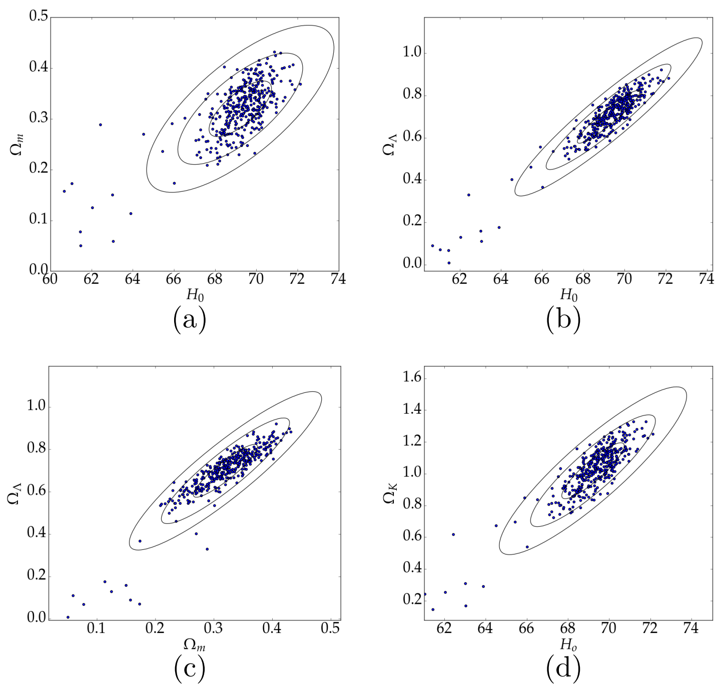

Figure 7.2: Confidence Region of parameters in $1\sigma, 2\sigma$ and 3σ contours. (a)-(Ω_m, H_0), (b)-(Ω_Λ, H_0), (c)-$(\Omega_\Lambda, \Omega_m)$ and (d)-(Ω_k, H_0)

Obtained mean values of parameters from Markov chains are

$Parameter$	$Mean$	σ
Ω_m	0.29	± 0.012
Ω_Λ	0.69	± 0.033
$H_0(km/s/Mpc)$	71.23	± 1.43

7.2 Parameter estimation of flat ΛCDM model with parameter set $p = (H_0, \Omega_{m0})$

Data used for estimation is same we used in previous one. For a flat universe $k = 0$, So Eq 6.28 is fitted.Estimated parameters from Markov chain are:

$Parameter$	$Mean$	σ
Ω_m	0.28	± 0.012
$H_0(km/s/Mpc)$	70.98	± 0.95

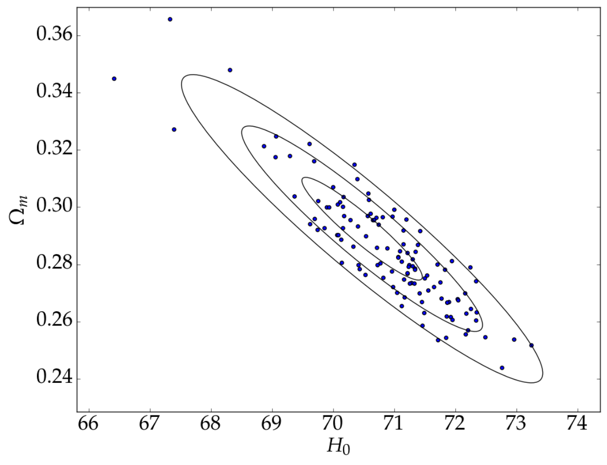

Figure 7.3: Confidence region of Ω_m and H_0

7.3 Conclusion

In this dissertation we present our calculations of hubble's constant and present cosmological density parameters.For that we developed a computational program to generate parameter chains using Markov Chain Monte Carlo algorithm. Implemented multi threading method for chain generations with Gelman Rubin diagnostics as convergence alarm. Program is tested with different multi parameter functions. Data of OHD [7][9][12][13][16][17] are used to find cosmological parameters,from our calculations we found that, for a curved universe $\Omega_m = 0.29 \pm 0.012$, $\Omega_\Lambda = 0.69 \pm 0.033$ and $H_0(km/s/Mpc) = 71.23 \pm 1.43$. For a flat universe $\Omega_m = 0.28 \pm 0.012$ and $H_0(km/s/Mpc) = 70.98 \pm 0.95$.

Bibliography

[1] *Introducing Monte Carlo Methods with R.* Springer Science+Business Media, LLC, 2010.

[2] Herman Bruyninckx. Bayesian probability, 2002.

[3] Y. Chen and B. Ratra. Hubble parameter data constraints on dark energy. *Physics Letters B*, 703:406–411, September 2011.

[4] www.tutorialspoint.com cpp multithreading.

[5] A. Heavens. Statistical techniques in cosmology. *ArXiv e-prints*, June 2009.

[6] https://en.wikipedia.org.

[7] Raul Jimenez, Licia Verde, Tommaso Treu, and Daniel Stern. Constraints on the equation of state of dark energy and the Hubble constant from stellar ages and the CMB. *Astrophys. J.*, 593:622–629, 2003.

[8] Dirk P. Kroese. Monte carlo methods. *University of Queensland*, 2011.

[9] Michele Moresco, Licia Verde, Lucia Pozzetti, Raul Jimenez, and Andrea Cimatti. New constraints on cosmological parameters and neutrino properties using the expansion rate of the Universe to z 1.75. *JCAP*, 1207:053, 2012.

[10] Burno A. Olshausen. Bayesian probabality theory. *1574.01497v1*, 2004.

[11] C. P. Robert. The Metropolis-Hastings algorithm. *ArXiv e-prints*, April 2015.

[12] Ariel G. Sanchez, M. Crocce, A. Cabre, C. M. Baugh, and E. Gaztanaga. Cosmological parameter constraints from SDSS luminous red galaxies: a new treatment of large-scale clustering. *Mon. Not. Roy. Astron. Soc.*, 400:1643, 2009.

[13] D Stern, Licia Verde, J Simon, and Raul Jimenez. physics. *JCAP*, 2008.

[14] L. Verde. Statistical Methods in Cosmology. In G. Wolschin, editor, *Lecture Notes in Physics, Berlin Springer Verlag*, volume 800 of *Lecture Notes in Physics, Berlin Springer Verlag*, pages 147–177, March 2010.

[15] Licia Verde. A practical guide to Basic Statistical Techniques for Data Analysis in Cosmology. 2007.

[16] Licia Verde, J Simon, and Raul Jimenez. Optimizing CMB polarization experiments to constrain inflationary physics. *JCAP*, 0601:019, 2005.

[17] C. Zhang, H. Zhang, S. Yuan, S. Liu, T.-J. Zhang, and Y.-C. Sun. Four new observational H(z) data from luminous red galaxies in the Sloan Digital Sky Survey data release seven. *Research in Astronomy and Astrophysics*, 14:1221–1233, October 2014.

Index

APPENDIX - I

Program Code

The Multi-thread Algorithm is developed on C++ and comprises of about 600 lines. Python language is used for data analysis and plotting and this code is also about 200 lines. To lessen out the bulkiness of the Project Report, actual project code has been uploaded to Github code repository and is available freely to view and analyze at https://github.com/antolonappan/Project-code