Exploring the MCMC for parameter estimation

This project is accomplished with affiliation to
Ahmedabad University

As part of the PhD coursework for the course PHY798 - Research Project - II

By,
Kishan Malaviya
AU2329011

Under the guidance of
Professor Gaurav Goswami
Mathematical and Physical Sciences Division,
Ahmedabad University

Ahmedabad
University

Contents

1 Introduction

Imagine a scenario where you have some observed data based on an experiment that you performed. You wish to understand the science behind this experiment. From your knowledge, you came up with a theoretical model that is capable of explaining the data. The model takes a few parameters as input to describe any physical scenario. In order to explain the observed data using your theoretical model, you need to know the appropriate input parameters to your model. Now this is a big problem because there can be numerous different possible combinations of parameters and you have to check each of them to know which combination explains your data well. With the help of the MCMC algorithms, one can make this problem much easier to solve.

Given a model and data, the MCMC algorithms can give us the most probable values of parameters for the model to fit the data. Note that, the MCMC is not a fitter but a sampler. That is, MCMC won't give you the best fit parameters but it'll give you the marginal distributions of the parameters through which can find the most probable value of each parameter in your model.

In this document, we give a brief about what MCMC is and what it does. We explain one of the most basic MCMC algorithms : The Metropolis - Hastings Algorithm and its implementation. At the later part, we discuss the application of MCMC in the analysis of the rotation curves of the galaxies for estimating the model parameters describing the Ultra Light Dark Matter (ULDM). This document is supposed to be a beginner level introduction to the MCMC and how it is implemented. Thus, the reader is encouraged to read from any standard textbook on MCMC to get better understanding of the concepts discussed below.

2 "Monte Carlo" and "Markov chains"

2.1 The Monte Carlo Methods

Consider, a random variable X which follows a known distribution p(x) and we wish to compute the expected value of some function of X (say $f(X)$). This is given by

$$E[f(X)] = \int f(x)p(x)dx$$

In many cases, this integral can be very difficult to be solved analytically as a formula for the indefinite integral may not be available in closed form. In these type of cases, one can estimate the integral by taking random independent samples from the distribution of X i.e. p(x) to generate 'n' random samples :

$$x_1, x_2, x_3, \ldots, x_n$$

then one can compute the following :

$$S_n = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

when the n is sufficiently large S_n approximates the $E[f(X)]$ in accordance with the law of large numbers. The error in this case would be of the order of $\mathcal{O}\left(n^{-\frac{1}{2}}\right)$. Note that, the random variable X could be taking value in $\mathbb{R}^d$ for any dimension d. The key point to note here is that the error is independent of the dimension. The class of methods where we generate samples $x_1, x_2, x_3, \ldots, x_n$ to compute quantities related to the distribution of X are called the **Monte Carlo Methods** or simulation.

In summary, The aim of the monte carlo methods are to solve one or both of the following problems.

1. To generate samples from a given probability distribution $p(x)$
2. To estimate the expectation values of functions under the distribution $p(x)$

In the Monte Carlo methods, how we take samples (i.e. method of sampling) from $p(x)$ is very important. There are various sampling methods available such as random sampling, importance sampling, MCMC sampling etc. Apart from that, The Monte Carlo methods work well when one knows the $p(x)$ as sampling becomes easy with known $p(x)$. In cases where we do not know the $p(x)$, we first have to come up with a way to simulate the $p(x)$ through some way. In Markov Chain Monte Carlo (MCMC), we try to simulate a desired distribution using the Markov chains. Once we have the well approximated target distribution and samples from it, we can then use the Monte Carlo method to estimate the required quantity.

2.2 The Markov Chains

In the case of MCMC, one wants to simulate a desired distribution (target density) using Markov chains. To achieve this, we create a Markov Chain which eventually converges to our desired distribution. Here, we briefly discuss the concept of Markov chains and it's convergence.

The fundamental notions of Markov chains and some of the key results related to Markov chains are essential to establish the convergence of various MCMC algorithms. A Markov chain is a sequence of random variables $\Theta^{(0)}, \Theta^{(1)}, \Theta^{(2)}, \ldots$ evolving over time while following **the Markov property**. The Markov property makes each next step dependent only on the previous one step. Mathematically, It can be written as follows :

$$P(\Theta^{(t+1)}|\Theta^{(t)} = \theta^{(t)}, \Theta^{(t-1)} = \theta^{(t-1)} \ldots \Theta^{(0)} = \theta^{(0)}) = P(\Theta^{(t+1)}|\Theta^{(t)} = \theta^{(t)})$$

Here, the small theta (θ) denotes the value attained by the random variable Θ. The $\Theta^{(t)}$ can take any value from the state space S which is essentially the real interval defined by the allowed range of the parameter.

The Markov Property allows the Markov chains to be memoryless as each state depends only on the previous step. One can even form Markov chain within a different Markov Structure. A specific Markov structure refers to the dependence of a state $\theta^{(t)}$ on the previous states (Smith, 2021). For example, the first order Markov chain to the chain where the next state depends only on the previous step. Whereas, the second order Markov Chain referes to the chain where the next state depends on previous two states. In the case of MCMC, we deal with first order Markov Chain only.

A Markov Chain can be specified by :

1. **An initial state** - Initial probability distribution of each parameter
2. **The transition kernel** - Which allows movement from one state to another

As an example, consider a Markov chain with three states (s = 3). The transition matrix for this example is

$$T = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0.1 & 0.9 \\ 0.6 & 0.4 & 0 \end{bmatrix}$$

If the probability vector for the initial state is $\mu(\theta^{(1)}) = (0.5, 0.2, 0.3)$, it follows that $\mu(\theta^{(1)})T = (0.2, 0.6, 0.2)$ and, after several iterations (multiplications by T), the product $\mu(\theta^{(1)})T^t$converges to $p(\theta)$ = (0.2, 0.4, 0.4). No matter what initial distribution $\mu(\theta^{(1)})$ we use, the chain will stabilize at $p(\theta)$ = (0.2, 0.4, 0.4).

For any starting point, the chain will convergence to the invariant distribution $p(\Theta)$, as long as T is a stochastic transition matrix that obeys the following properties:

- **Irreducibility -** For any state of the Markov chain, there is a positive probability of visiting all other states. That is, the matrix T cannot be reduced to separate smaller matrices.
- **Aperiodicity -** The chain should not get trapped in cycles.

A sufficient, but not necessary, condition to ensure that a particular $p(\Theta)$ is the desired invariant distribution is the following reversibility (Also called - **the detailed balance**) condition (Andrieu et al., 2003)

$$p(\Theta^{(t)})T(\Theta^{(t-1)} \mid \Theta^{(t)}) = p(\Theta^{(t-1)})T(\Theta^{(t)} \mid \Theta^{(t-1)}).$$

Summing both sides over $\Theta^{(t-1)}$, gives us

$$p(\Theta^{(t)}) = \sum_{\Theta^{(t-1)}} p(\Theta^{(t-1)})T(\Theta^{(t)} \mid \Theta^{(t-1)}).$$

In continuous state spaces, the transition matrix T becomes an integral kernel K and $p(\Theta)$ becomes the corresponding eigenfunction

$$\int p(\Theta^{(t)})K(\Theta^{(t+1)} \mid \Theta^{(t)})\, d\Theta^{(t)} = p(\Theta^{(t+1)}).$$

The kernel K is the conditional density of $\Theta^{(t+1)}$ given the value $\Theta^{(t)}$.

When the convergence to the invariant or stationary distribution is achieved the detailed balance condition holds, which shows that the probability to move from $\Theta^{(t-1)}$ to $\Theta^{(t)}$ is equal to the probability to move from $\Theta^{(t)}$ to $\Theta^{(t-1)}$. In the case of MCMC, one creates a kernel which creates a Markov Chain whose stationary distribution is our desired distribution.

3 The Markov Chain Monte Carlo (MCMC)

In the practical scenarios, we will have some data and a model which we would like to use to explain the data. The question of our interest would be : What is the most suitable set of parameters which can best-fit my model on the data ? In this case, The probability of our model fitting our data ($p(\Theta|D)$) can be given by the Bayes Theorem as follows :

$$p(\Theta|D) = \frac{p(D|\Theta)p(\Theta)}{p(D)} \tag{1}$$

where ,

- $\Theta = \{\theta_i\}$ is vector of our parameters
- $p(\Theta|D)$: probability of model given data (i.e. Posterior)
- $p(D|\Theta)$: probability of data given model i.e. how likely the data is given a model (Likelihood)
- $p(\Theta)$: probability distribution of our parameters (i.e. prior distribution)
- $p(D)$: The probability of our data (i.e., evidence)

The $p(D)$, in general, can be very complex analytically; hence, computing it can be computationally very expensive. Hence, we first need to simulate the posterior distribution from which we can sample. The general goal of the MCMC algorithms is to draw N samples $\{\theta_i\}$ from a well approximated posterior probability density $p(\Theta|D)$ (Foreman-Mackey et al., 2013).

An Markov Chain Monte Carlo (MCMC) algorithm to simulate a desired distribution P is any algorithm that can generate a Markov chain $\{\theta^{(t)}\}$, whose stationary distribution is P. In the context of Bayesian Analysis, the desired distribution P is the posterior $p(\Theta|D)$ so that $\{\theta^{(t)}\}$ can be treated as draws from $p(\Theta|D)$.

The most widely used MCMC algorithm is Metropolis-Hastings Algorithm (MHA). In MHA, we create random samples from the Markov chain specified by a combination of a proposal density ($q(\Theta^{(t)}|\Theta^{(t-1)})$)and Metropolis-Hastings Acceptance Ratio (a).

$$a(\theta^{(t-1)}, \theta) = \frac{\pi(\theta|x)\, q(\theta^{(t-1)}|\theta)}{\pi(\theta^{(t-1)}|x)\, q(\theta|\theta^{(t-1)})}$$

where π represents the posterior distribution.

If $a \geq 1$ then the new state is accepted.
Otherwise, the new state is accepted with probability a.
If the step is accepted, we set $\theta^{t+1} = \theta$.
If the step is rejected, then we set $\theta^{t+1} = \theta^{(t)}$

This acceptance ratio is inspired by the detailed balance condition required for the convergence of the markov chain. In MHA, the combination of acceptance ratio and proposal density act as the transition kernel for the markov chain. Also, note that by accepting the new parameters with a certain probability (a), we can avoid getting stuck around local maxima of the desired distribution.

Algorithm 1 The Metropolis-Hastings Algorithm

```
Initialize $\theta^{(0)}$
For $t = 1, \dots, N$
    1. Generate a candidate value $\theta \sim q(\theta|\theta^{(t-1)})$
    2. Compute the MH acceptance probability
       $\alpha(\theta^{(t-1)}, \theta) = \min\left\{1, \frac{\pi(\theta|x)\, q(\theta^{(t-1)}|\theta)}{\pi(\theta^{(t-1)}|x)\, q(\theta|\theta^{(t-1)})}\right\}$
    3. Set $\theta^{(t)} = \theta$ with probability $\alpha(\theta^{(t-1)}, \theta)$
       Otherwise set $\theta^{(t)} = \theta$
```

To perform MHA, We do the following :

1. we **start by an arbitrarily chosen point** in the parameter space.

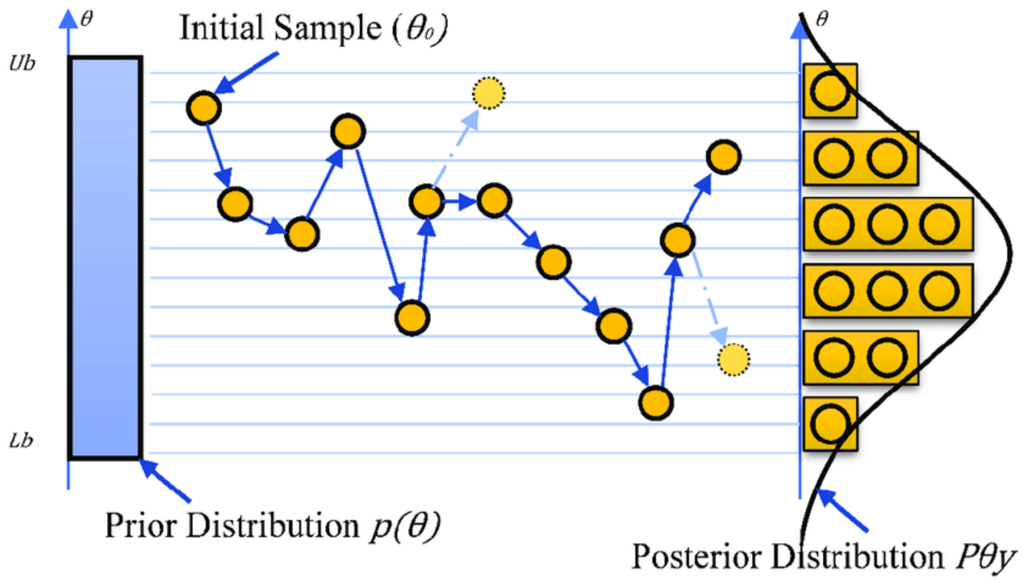

Figure 1: Graphical Representation of MCMC algorithm. Here,dark yellow circles represent accepted samples whereas faint ones represent rejected samples by the acceptance ratio. When chain is long enough and reaches convergence, the relative frequency distribution of the accepted samples appropriately approximates to the desired distribution (here, the Posterior distribution) (Credit - Milad et al. (2021))

2. We provide the initial set of parameters to our proposal density function whose job is to **propose new set of parameters** based on the current ones.

3. Once the new set of parameters are proposed. We calculate the Metropolis-Hastings Acceptance ratio (α). Then we **accept the new parameters with the probability** $min(1, \alpha)$.

4. We **repeat the steps-2 & 3 sufficient number of times** until the convergences is reached.

4 Implementing the Metropolis Algorithm

We have discussed the Metropolis Algorithm in the previous section. Now, We will see how we can implement these ideas as a code. The codes used in this document can be found at the GitHub repository - **MCMC-from-scratch**. We first start by creating functions which are necessary to implement the Metropolis Algorithm. Later, we will discuss the code for the the metropolis algorithm.

4.1 The necessary functions for MCMC

1. **A function for model :**
   This function encodes the information about our model. It takes the parameters of the model as input. For example, if my model is an equation of line, then the model function will look like the following :

   - : Code : -

```python
#defining the model
def model(x,params):
    m,c = params
    return m*x + c
```

2. **A likelihood function :**
   The Likelihood function tells how likely the data is for a given set of parameters for the given model. One typically uses the χ^2 - likelihood which is given by the following equation :

$$log(likelihood) = -\frac{1}{2}\left(\frac{y - model(params, x)}{error}\right)^2$$

   - : Code : -

```python
#Defining the likelihood
def log_likelihood(theta,x,y,yerr):
    #Here, x and y and err comes from the data
    return -0.5*np.sum(((y - model(x,theta))/yerr)**2)
```

3. **A prior distribution :**
   This function provides the prior distribution corresponding to the parameters involved in the model. Usually, one picks uniform prior distribution. This function is helpful in ensuring that our walker explore

only a certain region of interest in the parameter space. For example, consider the following prior distribution (log_prior(params)). Here, if m is between 0 and 5 and c is between -3 and 3, the function returns a constant (Here, 0). For any value of m and c outside of this ranges, it returns $-\infty$, excluding their contribution.

- : Code : -

```python
#Defining the prior
def log_prior(params):
    #This function ensures that the parameters come from
   our desires range
    m,c = params
    if (0<m<5) and (-3<c<3):
        return 0
    return -np.inf # -infinity for unallowed values
```

4. **A function for posterior :**
The posterior function is taken as per the Bayes' Theorem as follows :

$$log(posterior) = log(likelihood) + log(prior) - log(evidence)$$

Here, the $log(evidence)$,i.e. $P(D)$ is a constant as it doesn't depend on the parameters of the model, so we neglect the last term in the above equation,

$$log(posterior) = log(likelihood) + log(prior)$$

Here, we also impose the condition that if $log(prior(params))$ returns a value which is not finite then the function returns $-\infty$, excluding the contribution.

- : Code : -

```python
# Define the posterior as a combination of likelihood and
    prior
def log_posterior(x,y,err,params):
    lp = log_prior(params)
    if not np.isfinite(lp):
        return -np.inf
    return log_likelihood(params,x,y,err) + log_prior(
   params)
```

5. **The proposal distribution :**
   The proposal density function is one of the most important function in the MCMC algorithm. The job of this function is to propose new set of parameters given current parameters. The following is an example of a proposal function for the the example where we estimate the parameters of a line (i.e., m and c) using MCMC. Here, a normal distribution of a user specified width(σ) is set around the current value of parameter and a random value is proposed based on this normal distribution.

   - : Code : -

```python
# defining the proposal distribution, here proposal distribution is gaussian / normal
def proposal_density(current_params, proposal_params):
    current_m, current_c = current_params
    proposal_width_m, proposal_width_c = proposal_params

    proposed_m = np.random.normal(current_m, proposal_width_m)
    proposed_c = np.random.normal(current_c, proposal_width_c)

    return proposed_m, proposed_c
```

4.2 The Metropolis Algorithm

Now that we have all the functions ready, we can implement the Metropolis Algorithm. Recall the proposal density function defined in the previous section, note that this is a symmetric proposal distribution; that is, the probability of going from X to Y is same as that of going from Y to X. If the proposal density is symmetric then the Metropolis - Hastings algorithm is referred as the Metropolis Algorithm. Note that, in the case of the Metropolis Algorithm, the acceptance ratio is just the ratio of posterior distributions at $\theta^{(t)}$ and $\theta^{(t-1)}$, respectively. Here, The proposal density combined with the acceptance ratio acts as the transition kernel for generating the Markov Chain.

- : Code : -

```python
    # The Metropolis-Algorithm MCMC

```

```python
def metropolis_hastings(x,y,err,initial_params,
    proposal_params,n_iters):
    # Initial values
    current_m,current_c = initial_params
    trace_params = [[current_m],[current_c]]


    for i in range(n_iters):

        # Propose new values from current_values based on the
     proposal density
        current_params = current_m,current_c
        proposed_params = proposal_density(current_params,
    proposal_params)

        # Calculate log-posterior for current and proposed
    values
        current_log_posterior = log_posterior(x,y,err,
    current_params)
        proposed_log_posterior = log_posterior(x,y,err,
    proposed_params)

        # Acceptance ratio (The metropolis acceptance ratio)
        acceptance_ratio = np.exp(proposed_log_posterior -
    current_log_posterior)

        # Accept or reject the new state
        u = np.random.rand()
        if u < acceptance_ratio:
            current_m = proposed_params[0]
            current_c = proposed_params[1]

        # Append the current values to the trace
        trace_params[0].append(current_m)
        trace_params[1].append(current_c)

    return np.array(trace_params)
```

The above function returns a numpy array called "**trace_params**", which contains the numpy arrays of samples collected for each of the parameters. One can visualize the convergence of the Markov chains by plotting the trace curves of the samples of each parameters. The relationship between the parameters can be visualized by plotting the marginal and join distributions for the parameters.

4.3 Other MCMC algorithms

The Metropolis-Hastings algorithm is the simplest and most widely used MCMC algorithm. There are many improved MCMC algorithm where the Markov Chain can converges faster than MHA, however, each algorithms have their shortcomings. Interested readers can read more about various MCMC algorithms at the Github - webpage by Michael Clark via this **link**. For this document, we here show the comparison between Metropolis-Hastings sampler and The Affine Invariant Ensemble Sampler (AIES) (Goodman and Weare, 2010). The AIES is the algorithm used by the **emcee - python package** for MCMC implementation (Foreman-Mackey et al., 2013). The emcee is widely used in the field of Astronomy and Cosmology owing to its faster and efficient convergence.

	The Metropolis Algorithm	**Affine Invariant Ensemble Sampler (AIES)**
Proposal	Isotropic random walk based on $\mathcal{N}(0, \sigma)$.	Proposals depend on the positions of other walkers.
Efficiency	Poor for high-dimensional or correlated distributions.	Efficient for anisotropic or correlated distributions.
Scalability	Single walker; not parallelizable.	Ensemble of walkers; highly parallelizable.
Affine Invariance	No affine invariance.	Invariant under affine transformations.
Convergence Speed	Slower, especially for correlated distributions.	Faster due to efficient exploration of parameter space.

Table 1: Comparison between The Metropolis Algorithm and Affine Invariant Ensemble Sampler

5 Analysing the rotation curves

The rotation curve of a galaxy is plot of orbital velocity of stars and gas as a function of their radial distance from the center of the galaxy. The total observed rotational velocity can be broken down into two constituent parts : Baryonic (stars in the disk & bulge along with gas in the galaxy) and Dark matter. Mathematically,

$$V_{obs} = \sqrt{V_{DM}^2 + V_g|V_g| + \Upsilon_d V_d|V_d| + \Upsilon_b V_b|V_b|}$$

where V_d, V_b and V_g are contributions from stellar disk,bulge and gas components of the galaxy, while V_{DM} is the dark matter contribution. In the bulge-less dwarf galaxies, the contribution from the bulge is zero i.e, $V_b = 0$ at all radial distances, whereas the contributions from the stellar disk and gas is also small compared to the dark matter owing to the small size of the galaxy.

The rotation curves are one of the most important tool for studying the dark matter in the galaxies. Using the rotation curve along with the Virial Theorem, one can estimate the dynamical mass of the galaxy which is the sum of total visible (i.e. Baryonic) mass and the mass contained in the form of dark matter. Here, we show the application of MCMC in the context of estimating the dark matter properties through rotation curve data from the Spitzer Photometry & Acuurate Rotation Curves (SPARC) catalog (Lelli et al., 2016) which contains high quality HI/Hα rotation curves for sample of 175 galaxies. For illustration, we use the rotation curve of UGC5721 dwarf galaxy for the MCMC analysis.

MCMC for ULDM without self interaction

From the Newtonian Mechanics, one can write the circular velocity of a test particle moving under the influence of a spherically symmetric density profile in the following manner :

$$v(r) = \sqrt{\frac{GM(r)}{r}} = \sqrt{\frac{4\pi G \int_0^r \rho_{DM}(r')r'^2 dr'}{r}}$$

For dwarf galaxies like UGC5721, the dominant component is DM hence we have used $\rho_{total} = \rho_{DM}$. Here, we consider the dark matter to consist of ultralight spin-zero scalar particles with a mass of approximately $10^{-22}eV$. Simulations show that such ultralight dark matter exhibit a core-halo structure. In this structure, the inner regions of the halo are characterized by flat density cores, which are stationary-state solutions to the Schrödinger-Poisson system of equations. Beyond a certain transition radius (r_t), ULDM behaves similarly to cold dark matter (CDM), and its density profile can be described using the well-known Navarro-Frenk-White (NFW) profile (Dave and Goswami, 2024). Therefore, the total dark matter density profile for a galactic halo can be expressed as follows:

$$\rho_{DM}(r) = \rho_{ULDM}\Theta(r_t - r) + \rho_{NFW}\Theta(r - r_t)$$

where,

$$\rho_{\mathrm{ULDM}}(r) \simeq \frac{0.019 \times (m/10^{-22}\,\mathrm{eV})^{-2}(r_c/\mathrm{kpc})^{-4}}{[1 + 0.091 \times (r/r_c)^2]^8}\, M_\odot/\mathrm{pc}^3, \tag{2}$$

where r_c is defined as the radius at which the density becomes half its central value, and is given by

$$r_c = 0.8242\left(\frac{s}{10^4}\right)\left(\frac{m}{10^{-22}\,\mathrm{eV}}\right)^{-1}\ \mathrm{kpc}. \tag{3}$$

Note that the free parameters here are the ULDM particle mass m and the scale parameter s.

Similarly, The NFW density profile is given by

$$\rho_{\mathrm{NFW}}(r) = \frac{\rho_s}{\frac{r}{r_s}\left(1 + \frac{r}{r_s}\right)^2}\, M_\odot/\mathrm{pc}^3. \tag{4}$$

Here ρ_s and r_s are halo-specific parameters. Since we impose continuity at the transition radius, i.e. $\rho_{ULDM}(r_t) = \rho_{NFW}(r_t)$, we can eliminate ρ_s and describe the NFW density using only r_t and r_s. Hence, we are left with 5 free parameters describing the rotation curves : ULDM particle mass (m), the scale parameter (s), the transition radius (r_t), the scale radius of the NFW profile (r_s) and the stellar mass-to-light ratio Υ_d.

Now, our goal is to identify the what should be the values of these 5 parameters to get best-fit of our model on the observed data of the rotation curve. To do that, we follow the procedures mentioned in the section 4.1 and 4.2 for this with model function incorporating the above discussed theoretical model of the ULDM without self interaction. We can validate our estimated parameters for the UGC5721 by comparing it with the parameters estimated by Dave and Goswami (2024).

The following is the best-fit plot generated by running the MHA with the above discussed model on the observed SPARC data for the UGC5721. Here, we have considered 100,000 iterations for the MCMC run and discarded first 60,000 samples as ’burn-in’ period (burn-in period is the part of chain till which the convergence is not achieved).

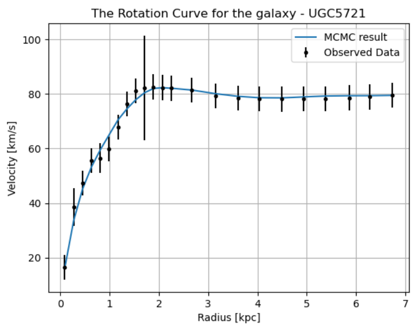

Figure 2: The MCMC fit for the observed rotation curve for the UGC5721

We can also visualise the markov chain by plotting the trace plots for each parameter. The following are the trace plots (Figure - 3) for the run. When the chain converges, it new values go back and forth around a certain value. To understand the relationship between the parameters, we can plot the covariance plots between the parameters. The following is the ”corner” plot (Figure - 4) generated for the obtained chain (burn-in is discarded).

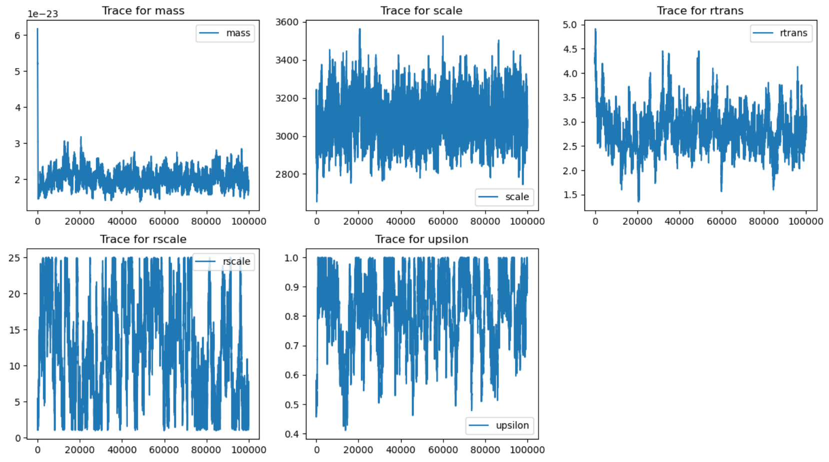

Figure 3: The trace plots of each parmeters showing the Markov Chain

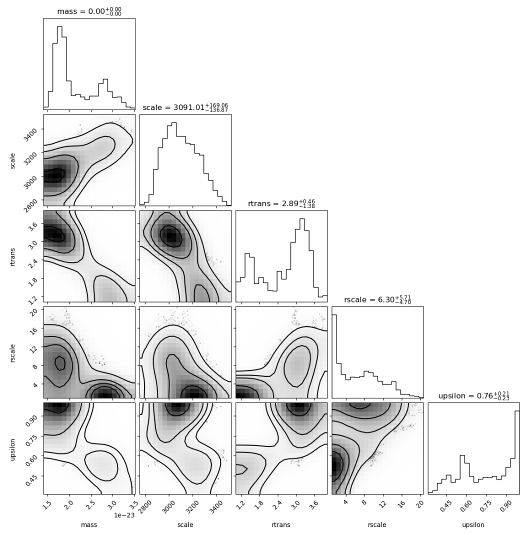

Figure 4: The corner plots for the obtained Markov Chain

6 Future Prospects

So far, we have used the Metropolis-Hastings algorithm with parameters which can attain any value from allowed intervals. However, when a parameter is discontinuous such that it can attain only certain discrete values from a given set of possible value, the proposal density has to be designed carefully. In this document, we have discussed the parameter estimation for ULDM without self interactions. In the case of ULDM with self interactions, we do not have an analytical solution to the $\rho_{ULDM}(r)$ but we can have a numerical solution corresponding to a self-interaction parameter (λ) (Dave and Goswami (2023)). For a set of different values for self interaction parameter (λ), we can have a set of numerical solutions. In this case, apart from the 5 continuous parameters ($m, s, r_t, r_s, \Upsilon_d$), we have one discrete parameter (λ). For such a discrete parameter, we wrote a discrete gaussian sampler which can draw samples from a set of discrete values.

- : Code : -

```python
#Discrete Sampler
def draw_sample_discrete(values,current_value,sigma=1.25):
    values = sorted(values)
    current_index = values.index(current_value)
    # Calculate weights based on the Gaussian-like distribution centered on the current value
    weights = []
    for value in values:
        weight = math.exp(-((values.index(value) - values.index(current_value)) ** 2) / (2 * sigma ** 2))
        weights.append(weight)
    weights = np.array(weights)
    # Normalizing the weights to get probabilities
    probabilities = weights / weights.sum()
    # Drawing a sample from values based on the probabilities
    sample = np.random.choice(values, p=probabilities)

    return sample
```

However, We found that when such a dicrete sampler is incorporated in the MHA, the Markov chain gets stuck around a local maxima of the simulated posterior distribution. This indicates that, in the case where the discrete and continuous parameters are involved simultaneously, a more sophisticated algorithms are required. Hence, one can explore an MCMC algorithm with an appropriate transition kernel where a walker can explore the parameter space of (discrete + continuous parameters) efficiently.

Further Reading

The discussion about MCMC in this document is based on the below provided list of resources. These resources will be helpful to anyone willing to delve deeper into specific concepts regarding the MCMC.

- MacKay, D. J. (2003). Information theory, inference and learning algorithms. Cambridge university press.
- Robert, C. P., Casella, G., & Casella, G. (1999). Monte Carlo statistical methods (Vol. 2). New York: Springer.
- Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (1995). Bayesian data analysis. Chapman and Hall/CRC.

References

Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to mcmc for machine learning. *Machine learning*, 50:5–43.

Dave, B. and Goswami, G. (2023). Self-interactions of uldm to the rescue? *Journal of Cosmology and Astroparticle Physics*, 2023(07):015.

Dave, B. and Goswami, G. (2024). Learning from galactic rotation curves: a neural network approach. *arXiv preprint arXiv:2412.03547*.

Foreman-Mackey, D., Hogg, D. W., Lang, D., and Goodman, J. (2013). emcee: the mcmc hammer. *Publications of the Astronomical Society of the Pacific*, 125(925):306.

Goodman, J. and Weare, J. (2010). Ensemble samplers with affine invariance. *Communications in applied mathematics and computational science*, 5(1):65–80.

Lelli, F., McGaugh, S. S., and Schombert, J. M. (2016). Sparc: Mass models for 175 disk galaxies with spitzer photometry and accurate rotation curves. *The Astronomical Journal*, 152(6):157.

Milad, A., Adwan, I., Majeed, S., Memon, Z., Bilema, M., Omar, H., Abdolrasol, M., Usman, A., and Md Yusoff, N. I. (2021). Development of a hybrid machine learning model for asphalt pavement temperature prediction. *IEEE Access*, PP:1–1.

Smith, A. N. (2021). Notes on bayesian computation and mcmc.