MONTE CARLO MARKOV CHAIN TUTORIAL

Doan T. , Parsons M., Suhail S.

ISYE 6644 Group Project Spring 2024

E–mail: *tdoan64@gatech.edu, sabasuhail@gatech.edu, mparsons31@gatech.edu*

SUMMARY: Monte Carlo Markov Chain (MCMC) simulations represent a cornerstone in the field of statistical computing, offering robust methodologies for sampling from complex probability distributions where traditional analytical approaches fall short. This tutorial provides a thorough exposition of MCMC simulations, encompassing their history, theoretical underpinnings, practical applications, and methodological considerations. We begin by elucidating the fundamental concepts of MCMC, including the mechanics of constructing Markov chains that converge to a target distribution. The tutorial emphasizes the practical importance of MCMC methods in diverse areas such as finance, statistics, machine learning, and beyond, illustrating how these techniques facilitate deep insights into data-driven problems.Key aspects of the tutorial include detailed descriptions of the most widely used algorithms such as the Metropolis-Hastings accompanied by code and numerical examples to guide the reader through their implementation.Through this tutorial, readers will gain a solid foundation in MCMC methodologies, empowering them with the skills to implement these techniques in various real-world settings. The tutorial aims to serve as a comprehensive guide for both novices looking to acquaint themselves with MCMC and seasoned practitioners seeking to refine their skills and explore advanced topics in the field.

1. Introduction

Monte Carlo Markov Chain (MCMC) simulations methods are powerful computational tools that combine the benefits of both the "randomness" of Monte Carlo simulations and "mechanics" of Markov Chains' state-dependent transitions. This fusion enables a way to explore complex probability distributions and complex problems that are unsolvable via analytical and numerical methods alone.

2. History and Theory

2.1. Where does Monte Carlo Markov Chain (MCMC) simulation originate?

The Monte-Carlo side of MCMC originated in the middle of the 20th century, spurred by the works of scientists such as Metropolis, Ulam, or von Neumann, who were involved in nuclear weapons projects, seeking solutions along the lines of particle distributions. Additionally, the Monte-Carlo method can be traced back even further to the works of both Count Buffon (who became famous for the Buffon's Needle Problem), as well as William Sealy Gosset (13 June 1876 – 16 October 1937), who was an English statistician, chemist and brewer (i.e., Head Brewer and Experimental Brewer at Guinness). In particular, Gosset became famous for creating the Student's T-Distribution. In more modern times, the widespread recognition of the use-case value of Markov chains was materially progressed by the work of Gelfand and Smith in 1990's. Even though there existed earlier scholarly contributions (such as those by Hastings in 1970, Geman and Geman in 1984, and Tanner and Wong in 1987) it was the works of Gelfand and Smith that elevated MCMC into the mainstream-statistical community. Initially, there existed a reluctance to embrace MCMC (in the mainstream-statistical community) due to the following factors: The primitive state of computational resources at the time (1970's); A general lack of familiarity with Markov chain theory; and scepticism regarding the method's practical utility.(13)

Gelfand and Smith pioneered efforts that played a crucial role in altering the perception that MCMC's lacked real-world practical utility. Gelfand and Smith achieved this by (with the support of their collaborators), publishing a series of papers that effectively demonstrated: The accessibility, implementability; and practicality of these methods (Gelfand et al., 1990; Gelfand, Smith, and Lee, 1992; Smith and

Gelfand, 1992; Wakefield et al., 1994).

Additionally, a secondary catalyst to the broader acceptance of MCMC methods was the introduction of BUGS (Bayesian inference Using Gibbs Sampling) software, which was birthed at the Valencia meeting in 1991. BUGS provided a powerful enablement to the practical adoption of MCMC algorithms on a wide scale.

2.2. The theory behind Monte Carlo Markov Chain (MCMC)

What is Markov Chain or Markov Process? It is a stochastic model describing possible events in which the probability of each event depends only on the state attained in the previous event. The below graphical model represents the Markov Chain in which:

$$P(X5|X4, X3, X2, X1) = P(X5|X4). \quad (1)$$

What is Monte Carlo Method? Given $X_1, X_2, ..X_n$ are independent and identically distributed , our goal is to calculate the expectation value of a function $g(x_i)$ in which computation using integration or exact numerical methods aren't available. Suppose that we can simulate $X_1, X_2, ..X_n$ i.i.d having the same distribution as X, we then can repeat the trials and observe $g(x_i)$,calculating its mean and variance. Based on the Central Limit Theorem (CLT), we know that if the sample size is large enough, the sample mean $\hat{\mu_n}$ will be an unbiased estimate for the true mean of function g(x). (1)

$$\hat{\mu_n} = \frac{1}{n}\sum_{n=1}^{n} g(x_i) \quad (2)$$

The variance in the CLT can be estimated by:

$$\hat{\sigma_n^2} = \sum_{n=1}^{n}(g(X_i) - \hat{\mu}_n)^2 \quad (3)$$

Due to the randomness in performing simulation or generating pseudo randomness, the theory of Ordinary Monte Carlo is elementary statistics in which we simply can not estimate a " point" or " point estimator" for the mean but rather an approximation or Monte Carlo calculation of the mean. (12)

So what is Monte Carlo Markov Chain (MCMC)? Monte Carlo Markov Chain is, therefore, an infusion of Monte Carlo in which $X_1, X_2..X_n$ is a stationary Markov chain having the initial distribution the same as the distribution of X. By sampling from the Markov Chain over multiple iterations, the algorithm eventually converges to its stationary distribution, which represents the steady state of the system. In its simplest form, a Markov Chain is a sequencing of random variables where future states (e.g., decisions) do not depend on historical states (e.g., past decisions). Rather, a future state depends entirely on the current state. This concept is known as the "memoryless" property. Interestingly, the thread that weaves current states with future states is based on transition matrices. Within these transition matrices, are elements that express the probability of state-transitions (i.e., the likelihood of transitioning from one state to another state). Future states are driven by these transition-probability elements.

2.3. What are the benefits of the Monte Carlo Markov Chain infusion and real world practical use cases?

One of the key benefits of infusing a Markov-chain methodology with Monte Carlo simulation methods is that the combined methodology is "smarter" than when each is used independently. For example, the "smartness" in MCMC comes from the Markov-chain component. The Markov-chain building block ensures the problem-space exploration is not entirely random. Rather, it follows a path where the next-choice point depends only on the current position. That is, the MCMC employs the memoryless property (as mentioned above). Therefore, history is meaningless with respect to future-path-selection sets. The real benefit of this property is that it allows MCMC to efficiently explore vast and complex probability landscapes via making transitions to new states based on probabilistic rules. Rules of which are designed to ensure, over time, the chain of samples reflects the true underlying distribution of the problem at hand.

The detailed balance condition is a fundamental result in MCMC theory. It states that for a Markov chain to converge to its stationary distribution, the transition probabilities between states must satisfy a balance equation. Mathematically, this condition ensures that the Markov chain is reversible, meaning that the probability of transitioning from state i to state j is equal to the probability of transitioning from state j to state i in the stationary distribution. This property is crucial for designing MCMC transition kernels that preserve the desired stationary distribution.The Metropolis-Hastings algorithm is a great example and is one of the most widely used MCMC methods based on the idea of reversibility.

MCMC has found material, practical value across many fields in the real world. For example, in climate research, MCMC can be utilized to model complex-climate systems, as well as enable improved prediction quality with respect to future-state-climate changes. This can be achieved through exploring various parameter spaces of climate models.

Regarding medicine, MCMC can be applied in Bayesian-based-data analysis of clinical trials, which could enable researchers to better approximate the

effects of treatments under conditions of uncertainty. MCMC's ability to to deal with complex, uncertain systems and to generate insights from such systems, has made it an indispensable tool for statisticians, scientists, and analysts worldwide. The following is a non-exhaustive list of potential use cases for the application of MCMC.

1. Bayesian Statistics and Data Analysis: MCMC is a cornerstone for performing Bayesian statistical analysis because it allows statisticians and researchers to update their hypothesis as more evidence becomes available. This can be particularly useful in fields such as epidemiology (e.g., where MCMC methods can be used for estimating disease-transmission rates and for evaluating intervention effectiveness, based on data that may fluctuate). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7549798

2. Climate Science: Weather is a complex system containing many factors, such as: temperature, humidity, wind speed and atmospheric pressure. Modeling all these factors can be an extremely challenging task due to its interconnected and stochastic nature. However, MCMC can be applied by following these steps:

   (a) Model Presentation: Define a Markov Chain model with each state representing a specific weather condition, such as: rainy, cloudy, sunny, snowy etc..

   (b) Transition Probabilities: Estimate the transition probability between different weather states based on historical data and meteorological knowledge. Record it in the transition matrix.

   (c) Monte Carlo Simulation: Using MC methods, simulating the weather system by iteratively transitioning from one state to another according to the estimated probability in the transition matrix.

   (d) Sampling: Collect a large number of simulated weather sequence samples over multiple iterations of the Markov chain.

   (e) Predicting: Analyze the samples to make predictions about future weather patterns.

   Scientists who utilize complex-climate models can harness MCMC methods for analysing the complexity that's generated from these models, as well as improve the predictive quality of likely future-climate scenarios. Given the magnitude of uncertainty and the quantity of variables involved in climate models, MCMC can help users of these models to better understand the probability of various outcomes (e.g., temperature increases, or changes in precipitation patterns).https://www.nasa.gov/solar-system/what-is-nasas-mars-climate-modeling-center

3. Machine Learning (ML) and Artificial Intelligence (AI):

   MCMC methods can be used in machine learning for Bayesian inference. This can foster prediction-quality improvement by enabling algorithms to better learn from their data (e.g., from training data). For example, natural language processing, image recognition, and recommendation systems, can all benefit from MCMC because these types of models must harness inferences from complex, uncertain data.

   For example, regarding recommender systems, by utilising MCMC these systems could be improved by better handling sparse data (noting users typically interact with small subsets of product catalogues). That's because MCMC methods could model the underlying distributions of user preferences and product-item characteristics, by sampling posterior distributions of those latent variables.Take a music platform like Spotify as an example, which would have a massive pallet of customer and music data. While Spotify would have a significant amount of data at the platform level, at the individual-user level, Spotify may not have sufficient data to work with for enabling meaningful song or album recommendations. This is particularly true for users that don't listen to music so often, or who have a limited range of playlists they listen to. In such cases how does Spotify enable meaningful recommendations given the data sparsity that may exist at the user level? One way to improve the quality of song/album recommendations could be to utilize MCMC. How so? By utilising the small amounts of data that do exist at the user level (i.e., the spare data) the user's latent (think hidden) variables (e.g., song preferences that are implied not explicitly advised) could be used as the ingredients for isolating other songs/albums the user may also like (based on other users' preferences). For example, the particular songs that are listened to by a single user will contain certain beats, rhythms, be of certain genres, etc. By using those limited data points as clues, Spotify (through the support of MCMC) could improve its ability to infer what other songs users may like. Those inferences would be expressed as probabilities (e.g., probability a user will like a certain song/album). Or, more precisely as the posterior distributions. The posterior distributions are essentially the educated predictions of what other music users may enjoy. Additionally, the more feedback users provide (e.g., likes/dislikes) the more refined and accurate the recommender system algorithm would become, because of the feedback loop and Bayesian nature of the underlying algorithmic mechanics.https://www.youtube.com/watch?v=Fd1TRv4ZLAs

In terms of recent developments in AI (e.g., generative AI), it will be interesting to see if companies utilize MCMC to enable user-generated-entertainment-content creation. For example, Sam Altman (of OpenAI) recently posted an example of a short-animated film on Twitter. This animation, while short in duration, demonstrated a level of quality that was on par with triple-A game studio productions (such as Microsoft Studios, Bioware, etc.).https://openai.com/sora

Extrapolating from the above example on Spotify, what if a small set of samples (i.e., sparse data) from a set of user prompts is all that's needed to spark the creation of short films in the near future? It may be that as the Generative AI learns from these small-user-prompt-enabled data samples (supported by MCMC perhaps), that progressive content development (generated by users) could be enabled by small but frequent iterations and experiments (i.e., prompt-suites). This is just an idea, but it will be interesting to see if, and or, how MCMC is used in future Generative AI developments.

3. MCMC implementation

In the early years, MCMC methods were primarily implemented through custom code developed by researchers for specific applications. Software tools were often limited in functionality and accessibility, requiring expertise in both programming and statistical theory. The publication of foundational papers on MCMC algorithms, such as the Metropolis-Hastings algorithm and Gibbs sampling, laid the groundwork for future software development efforts. In the 1990s, BUGS, initially developed by Adrian Smith and others, was one of the earliest software packages specifically designed for Bayesian inference using MCMC methods. The software, which used Gibbs sampling techniques, was widely adopted in the fields, such as: epidemiology, ecology, and social sciences for its ease of use and flexibility. The development of user-friendly software, libraries and packages such as Stan, PyMC, and emcee (also known as "the MCMC Hammer"), in the early part of the 2010s, has played a significant role in making MCMC more accessible to users. These libraries provide high-level interfaces and consist of pre-implemented algorithms, which allows users to specify their probabilistic models and to perform MCMC sampling with minimal coding efforts.

One of the most widely used packages today is PyMC3, which its development began in 2003 in an effort to make MCMC more accessible to the applied-scientist community. PyMC3 (a python package) was developed to help users define their stochastic models as well as to support users in the construction of Bayesian-posterior samples (via utilising MCMC). The PyMC3 package can be installed in conjunction with other essential packages, such as NumPy, Matplotlib, Pytables, and Scipy. An easy installation can be acheived via terminal as follows: easy_install pymc . For a full guidance on how to install the packages, please refer to:https://pypi.org/

There are two types of random variables in PyMC3, called stochastic and deterministic classes. Deterministic variables are often defined by a mathematical function that returns a value, given a set of values for other parameters. Interestingly, deterministic variables are sometimes referred to the "systemic" part of the particular model. In contrast, stochastic variables are defined by their probability distribution and it's often the case that even when the parameters of their parent variables are known, the actual values (of stochastic variables) remain uncertain.

Once all variables and priors for the parameters of interest are defined, we can fit the model with a Markov-Chain-Monte-Carlo algorithm, using the function MCMC(). To run the sampler, we call the MCMC object's sample() method, provide the number of iterations, burn-in length as well as the thinning interval, as desired. Regarding convergence, we can monitor for convergence by utilizing diagnostic tools (such as trace plots). Once the sampling process is completed, the model will produce the characteristic of its posterior distribution. We can then calculate point estimates such as mean, median, etc. Finally, by validating the fitted model (by comparing the prediction results against the true observations), we can adjust or fine tune the model's parameters in order to improve its prediction quality in future model iterations. The authors of the PyMC3 package provide an extensive website that offers a suite of MCMC-implementation examples in which to explore. That can be found here: https://www.pymc.io/projects/examples/en/latest/gallery.html (3)

4. Metropolis Hastings Algorithm

The Metropolis-Hastings algorithm is a Markov chain Monte Carlo (MCMC) method used for obtaining a sequence of random samples from a probability distribution for which direct sampling is difficult. This algorithm allows sampling from the distribution without knowing the full normalization constant, which is often the case in many applications, such as statistical physics, Bayesian statistics, and machine learning.

The core idea behind the Metropolis-Hastings al-

gorithm is to construct a Markov chain that has the desired distribution as its equilibrium distribution. The algorithm uses an iterative approach to generate a sequence of sample values, where each new sample is generated based on a proposed move from the current position (**?**). The acceptance of each proposed move is determined in a probabilistic way, depending on how likely the move is under the target distribution (relative to the current position). An example of pseudocode can be found from page number 850 Murphy, 2012, Machine Learning: A Probabilistic Perspective.(10)

While the Metropolis-Hastings algorithm is powerful, its performance is heavily dependent on both the choice of the proposal distribution and the nature of the target distribution. Tuning parameters and choosing an effective proposal distribution are crucial for achieving efficient sampling and fast convergence.

Key Features

Ergodicity: Markov chains that are constructed by the Metropolis-Hastings algorithm are ergodic. The means convergence to the target distribution will occur as the number of iterations approaches infinity (regardless of the starting point).

Flexibility: The proposal distribution q can be chosen to suit the particular problem at hand, the choice of which can affect the efficiency and convergence performance of the algorithm. For example, a poorly chosen q might result in many rejected moves, leading to slow convergence.

Applicability: The algorithm does not require any knowledge of the normalizing constant of the target distribution, making it particularly useful in fields like Bayesian inference where the posterior distributions are often known only up to a proportionality constant.(5)

5. Other Flavors of MCMC Algorithm

There are several other significant MCMC techniques that exist, each of which have their own unique approaches and use-case applications. A brief overview of some notable MCMC algorithms, are as follows:

Gibbs Sampling:

Description: Gibbs Sampling is a special case of the Metropolis-Hastings algorithm that's used

Fig. 1: Metropolis Hastings Algorithm

Algorithm 24.2: Metropolis Hastings algorithm

1 Initialize x^0 ;
2 **for** $s = 0, 1, 2, \ldots$ **do**
3 Define $x = x^s$;
4 Sample $x' \sim q(x'|x)$;
5 Compute acceptance probability

$$\alpha = \frac{\tilde{p}(x')q(x|x')}{\tilde{p}(x)q(x'|x)}$$

Compute $r = \min(1, \alpha)$;
6 Sample $u \sim U(0, 1)$;
7 Set new sample to

$$x^{s+1} = \begin{cases} x' & \text{if } u < r \\ x^s & \text{if } u \geq r \end{cases}$$

when the joint distribution is known, but the conditional distributions are easier to sample from. It involves updating one component of the state vector at a time, conditional on all other components.

Advantages: It does not require the tuning of parameters, such as step sizes, and its typically simpler to implement when the conditional distributions are known and tractable.

Limitations: It can be slow if the variables are highly correlated because it updates one component at a time.(4)

Hamiltonian Monte Carlo (HMC):

Description: HMC utilizes techniques from physics to generate an intelligent proposal mechanism. This is done by utilising the gradients of log probability that are used to guide the sampling process. It avoids the random walk behavior, and the slow exploration of state-spaces that often characterizes many MCMC methods. This is achieved by proposing moves to distant points that have high-acceptance rates.

Advantages: Efficient in exploring high-dimensional spaces due to proposals being informed by gradient information.

Limitations: Requires computation of gradients, which can be computationally expensive and limits its use to differentiable models.(2)

Slice Sampling:

Description: Slice sampling improves simple-random walks by considering the slice-level sets of the target distribution. It involves drawing a value (uniformly) from the region under the plot of a target density function at a particular point.

Advantages: It adapts automatically to the characteristics of the particular distribution and it typically requires minimal tuning.

Limitations: Its performance can degrade in higher dimensions or when faced with complex-dependency structures.(11)

Sequential Monte Carlo (SMC):

Description: Also known as particle filters, SMC methods involve a sequence of important sampling steps that are combined with a suite of resampling steps, which are designed to handle scenarios where the target distribution changes over time.

Advantages: Particularly useful for time-series analysis and dynamic modelling; adapts to evolving distributions.

Limitations: Can be computationally intensive; suffers from particle degeneracy (where weight becomes concentrated on few particles).(9)

Affine Invariant MCMC:

Description: Popularized by the "emcee" Python implementation, these algorithms use an ensemble of "walkers" (i.e., concurrent chains in the MCMC algorithm) that explore the target distribution, invariant to affine transformations of the space.

Advantages: Good for problems that are exposed to degenerate or correlated parameters; requires less tuning of the proposal distribution.

Limitations: May not be as efficient as gradient-based methods (like HMC) in some contexts. Each MCMC algorithm have specific strengths and are suited to particular types of problems, which are dictated by factors such as dimensionality, derivative availability, and correlation structures in the parameter space. The choice of the right MCMC method is crucial for efficient sampling and can significantly affect the convergence properties and computational efficiency of the analysis. (8)

Rao–Blackwellization MCMC:

Rao–Blackwellization [Casella et al 1996] is a statistical method used to improve the efficiency of estimators by reducing their variance. Applied within the Monte Carlo Markov Chain (MCMC) context, Rao–Blackwellization involves averaging over some of the components of the posterior distribution analytically rather than sampling them directly. This technique leverages the Rao–Blackwell theorem, which shows how conditioning on sufficient statistics can lead to an estimator with lower variance.

Advantages: By reducing the variance of estimators, Rao–Blackwellization can provide more accurate estimates using fewer samples. This can be crucial in applications where sampling is computationally expensive or when the convergence of the MCMC is slow.

Limitations: Rao–Blackwellization requires that certain conditional expectations or integrations can be computed analytically. This is not always possible, especially in highly complex or non-linear models where such integrations become infeasible.(5)

6. Practical Implementation of Metropolis Hastings Algorithm in Python

- **With Simulated Data**
  We used the description provided on https://www.youtube.com/watch?v=KmqTrm-bn8k to show a practical example (while abstract) on how to implement an MCMC using simulated data. For context, in this example we've been provided a target-probability distribution, however we don't know the target's distribution in its entirety. Rather, we only know the numerator of the target distribution, which we refer to as *f(x)*.

```
def f(x):
    if x >=1:
        return np.exp(-(x-1)^2) + np.exp(-(x-1)**2)
    else:
        return np.exp((x-1)/3) + np.exp((x-1)**3)
```

  Therefore, we need to identify the remaining components of the target distribution *p(x)*. Why? Because we want to be able to draw samples from this "complete" target distribution *p(x)*. This is a classic MCMC problem.

  The original distribution is defined as *p(x)* and we want to draw samples from this function *p(x)*. It's important to note that the "known" part of the target distribution (i.e., the numerator that we refer to as *f(x)*) is proportional to the original (or "complete") target function *p(x)*. Because

of this proportional relationship, if we can find an adjustment (or stretch) factor, this factor can be utilized to approximate the complete original function *p(x)*. We can achieve this by calculating a normalization constant. Please note: for this exercise, Wolfram Alpha was used to work out this normalisation constant.

To recap, our goal is to sample from the original function *p(x)* but we only have a portion of that function (i.e., the known component being *f(x)*). Therefore, we want to identify a normalization factor that we can use to approximate the "full version" of the original function *p(x)*. Once that is done, we will be able to draw random samples from this "now complete" function (which is our ultimate goal). More specifically, we will be in a position to calculate the expected value of random samples that are generated from our original function *p(x)*.

If we derive an expected value that's close to the true value, we will know our process has worked. That is, we want the expected value of the samples taken by utilising the Metropolis Hastings algorithm to be as close as possible to the expected value of the original function *p(x)*. For the example that follow, the true value (for reference) is: 0.27172

Finally, please note that we've also utilised the Acceptance-Rejection method with *N(0,3)* and *N(1,4)* to estimate the expected value of the distribution, as well.

The results are as follows:

True Value = 0.2717289367474419 (by Wolphram Alpha)

N(0,3) :

Expected Value = 0.33
Number of Samples = 6836 out of 1000000 iterations

N(1,4) :

Expected Value = 0.32
Number of Samples = 89884 out of 1000000 iterations

Metropolis Hastings Algorithm with *N(x_prev,4)* :

Expected Value = 0.28

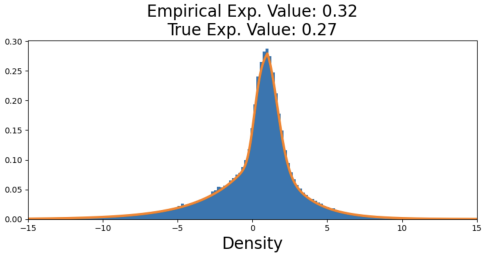


Fig. 2: Acceptance Rejection with N(1,4)

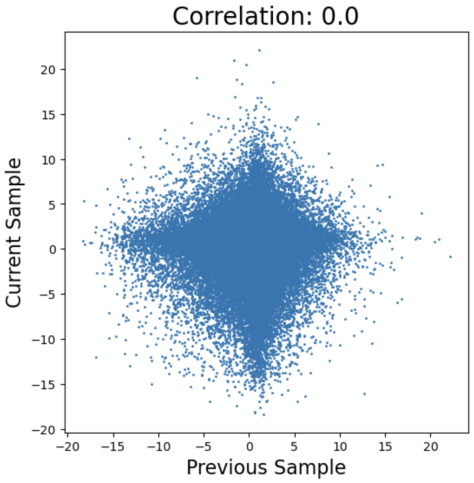


Fig. 3: Correlation in Acceptance-Rejection

Number of Samples = 999000 out of 1000000 iterations

Metropolis Hastings Algorithm with *N(x_prev,4* has the best efficiency and the best estimate of mean.(**?**)

- **With Oil Productivity Data**

  To extend our example to a more practical use-case, we've applied the MCMC methodology to an oil-productivity data set, which we found using this repository: https://github.com/michaelmml/Oil-Well-Productivity-SD. Here, the MCMC technique was used to help with the evaluation of oil-well productivity. In this particular example, oil productivity was assumed to follow a gamma distribution, and the MCMC sampling method was applied to create synthetic data. The visualised results are as follows:

  This example enhanced our understanding of the Metropolis-Hastings MCMC algorithm, emphasizing the importance of a well-crafted-model approximation that's developed in collaboration with subject-matter experts. It also demon-

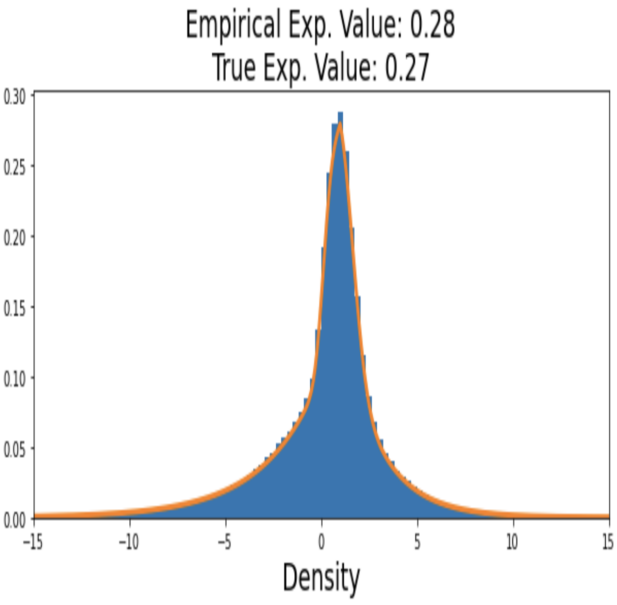


Fig. 4: Metropolis Hastings Algorithm with *N(x_prev,4)*

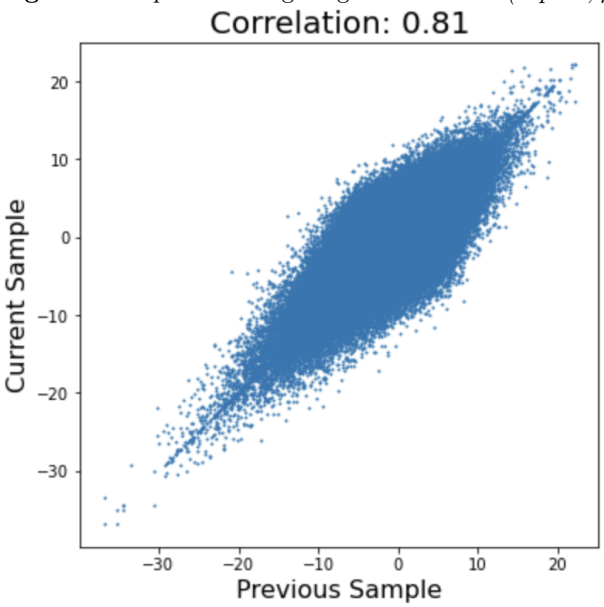
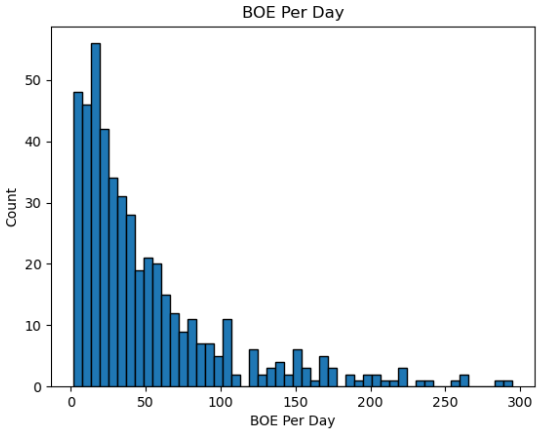


Fig. 5: Correlation in Metropolis Hastings Algorithm with *N(x_prev,4)*

Fig. 6: Original oil productivity data

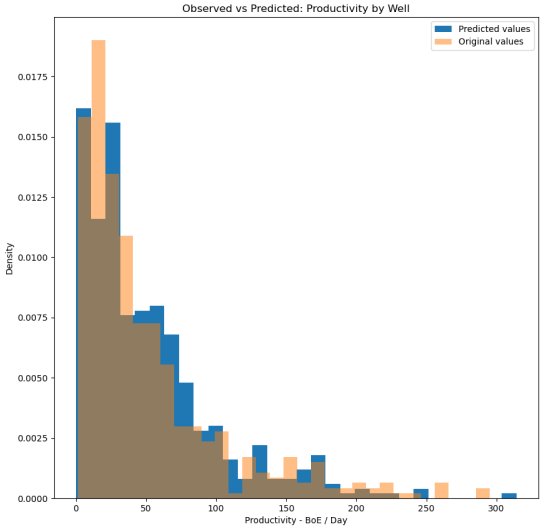


strates a strong foundational model not only facilitates more accurate sampling and convergence, but also ensures results are meaningful and applicable to the real-world under study. This insight also underscores the importance of having subject-matter experts be involved during the model-design phase of analytics projects such as this, and highlights how data-savvy experts, when combined with subject-matter experts, can offer real, tangible-insight value from models designed and developed collaboratively.

- **With Carbon Emissions Data**

As a final example showcasing a practical application of the MCMC methodology, we obtained carbon emissions data from major firms spanning various geographical locations worldwide (for the years 2011 and 2013). Given carbon emissions cannot be negative, our potential suite of approximate-distribution candidates were restricted to Weibull, Gamma, and Log-Normal distributions. Initially, we attempted to approximate by utilizing a Log-Normal distribution, but the results of our initial analysis did not support the use of Log-Normal distribution. Consequently, we explored the Gamma distribution instead, which was previously utilized in the oil productivity example. This offered much better results. In terms of process, we segmented our data for across the two years and computed both the alpha (shape parameter) and the beta (scale parameter) for each Gamma distribution, for each year. These parameters served as the ini-

Fig. 7: Comparison of real and simulated data for Oil Productivity Example



tial estimates for the Gamma distribution within the Metropolis-Hastings algorithm

For year **2011**:

Estimated shape parameter (alpha): 0.33860763598775423

Estimated scale parameter (beta): 16725298.297268819

For year **2013**:

Estimated shape parameter (alpha): 0.34964029771236194

Estimated scale parameter (beta): 16362447.98502256

The original emissions data for years 2011 and 2012 are shown in the first diagram above, while the comparison between the observed (actual) and predicted (simulated) data can seen in the second plot, above.

The estimates for alpha and beta, across years 2011 and 2013 for new data can be seen, below.

The code for all three examples, as well as the underlying data for examples two and three, are enclosed in the zip file titled Data and Notebooks.zip.

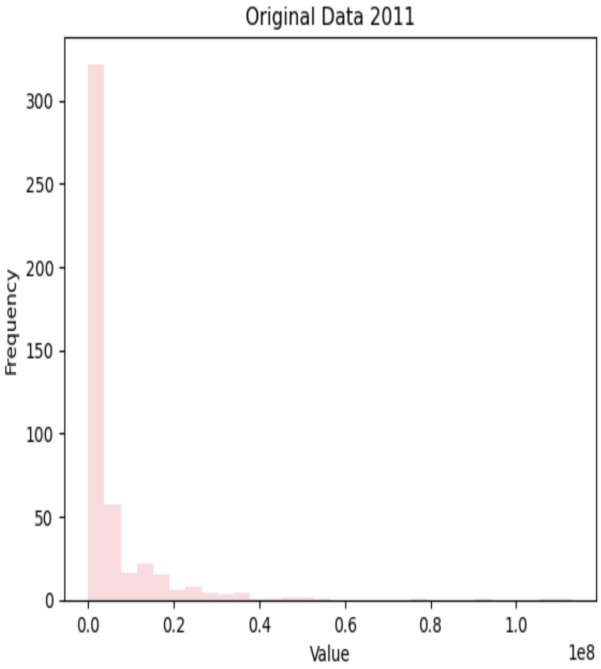


Fig. 8: Original Data 2011

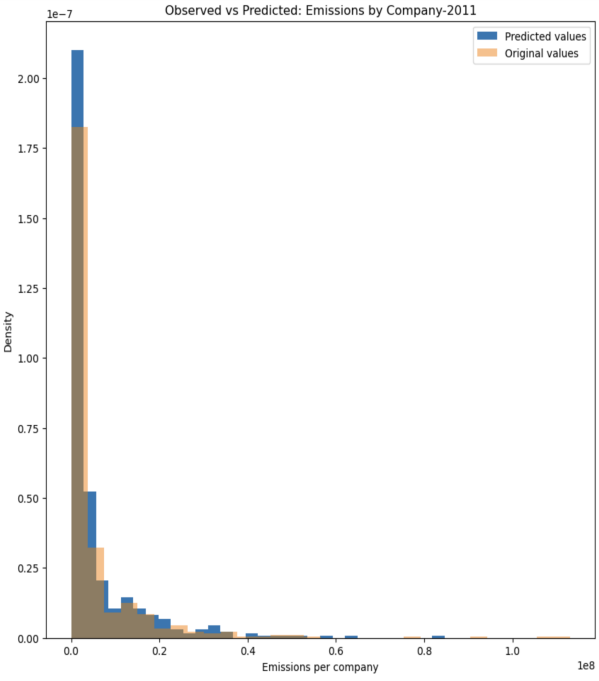


Fig. 9: Observed vs Predicted 2011

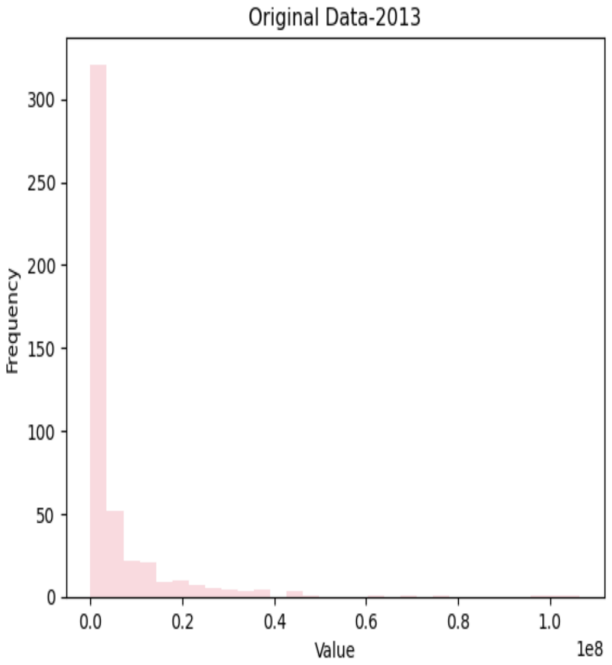


Fig. 10: Original Data 2013

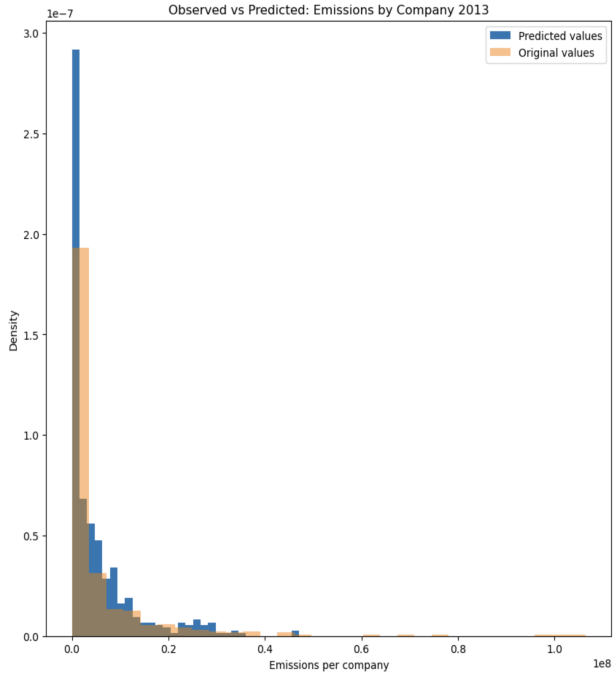


Fig. 11: Observed vs Predicted 2013

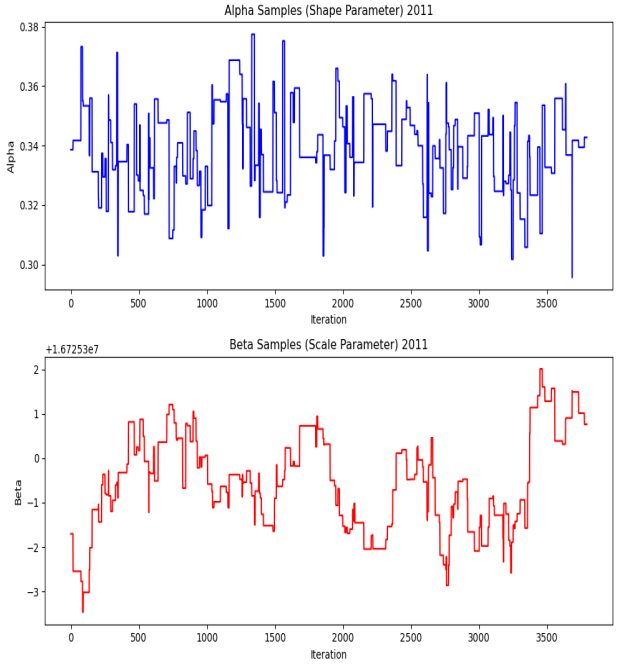


Fig. 12: Alpha and Beta updates-2011

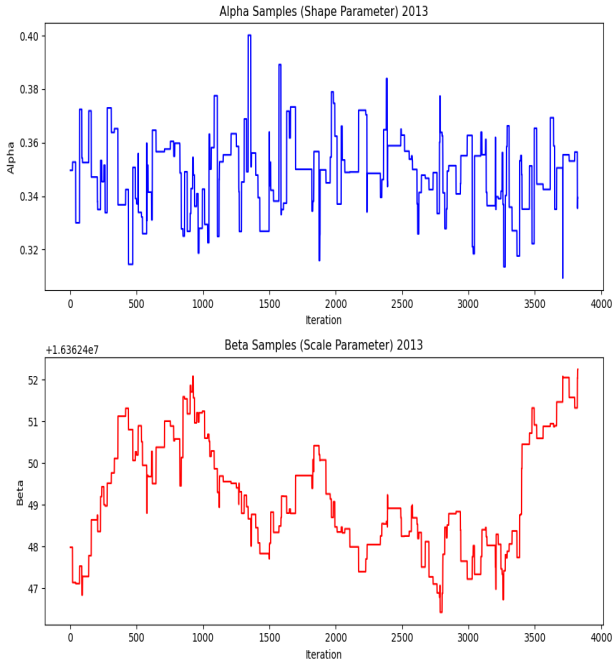


Fig. 13: Alpha and Beta updates-2013

Conclusion:

To conclude, the Monte Carlo Markov Chain (MCMC) method serves as a powerful statistical tool for analyzing complex probability distributions (that can otherwise be difficult to assess directly). By utilizing techniques such as sampling and stochastic processes, MCMC enables estimation of distribution parameters and model predictions, which can be applied across a diverse range of data sets.

In this paper, the utility of MCMC was illustrated through both abstract and practical examples, including the analysis of oil-well productivity and emissions data. These examples demonstrated MCMC's flexibility and effectiveness in tackling real-world problems where traditional analytical methods may fall short. By facilitating deeper insights into data characteristics and uncertainties, MCMC proves an indispensable tool for robust statistical inference in various scientific and engineering applications.

REFERENCES

[1] Markov Chain Monte Carlo (2023b, March 13). Columbia University Mailman School of Public Health. Mar 2023.

[2] M. Betancourt. A Conceptual Introduction to Hamiltonian Monte Carlo. Jan 2017.

[3] J. Brownlee. A gentle introduction to Markov chain Monte Carlo for probability. Sep 2019.

[4] G. Casella and E. George. Explaining the Gibbs Sampler. Feb 2012.

[5] G. Casella and Christian P. Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83(1):81–94, 03 1996.

[6] S. Chib and E. Greenberg. Understanding the Metropolis-Hastings Algorithm. , Sep 1995.

[7] Cassey P. Don, Van R. and .S Brown. A Simple instroduction to Markov Chain Monte Carlo Sampling, Mar 2016.

[8] C. J. Geyer and E. A. Thompson. Annealing Markov Chain Monte Carlo with Applications to Ancestral Inference. *Journal of the American Statistical Association*, pages 909–920, 1995.

[9] Doucet A. Moral, P. and A. Jasra. Sequential Monte Carlo Samplers. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 68, 2006.

[10] K. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, illustrated edition, 2012.

[11] Radford M. Neal. Slice sampling. *The Annals of Statistics*, 31(3):705 – 767, 2003.

[12] D. Patil A., Huard and C. J. Fonnesbeck. PyMC: Bayesian Stochastic Modelling in Python. *Journal of statistical software*, pages 1–81, 2010.

[13] G. Robert. C.P., Casella. A short history of markov chain monte carlo: Subjective recollections from incomplete data. *Statistical Science*, 1995.