Hamiltonian Monte Carlo

Zane Hassoun

November 21, 2021

1 Introduction

In this paper I will be discussing the Markov chain Monte Carlo (MCMC) method known as Hamiltonian Monte Carlo. MCMC is a method of simulation that was brought forward in a 1953 paper by Nicholas Metropolis to simulate ideal states of molecules . In 1959, the concept was extended by Alder and Wainright by creating a more deterministic outcome, which are now known as Hamiltonian Dynamics. The selected literature that popularized Hamiltonian Monte Carlo was developed by Duane, Kennedy, Pendleton and Roweth and published as "Hybrid Monte Carlo" and is col¬loquially referred to as "HMC." More recently, HMC has entered into the realm of Bayesian Statistics; in 1996 Neal published a paper applying HMC to neural networks To properly understand Hamiltonian Monte Carlo and Hamiltonian Dynamics, it is imperative to have a thorough foundational understanding of Monte Carlo simulations, Markov chains, Markov chain Monte Carlo (MCMC) methods and various supporting algorithms such as, Metropolis-Hastings, Random Walk etc). [3].

2 Foundations

2.1 Markov chain Monte Carlo

Because there are two parts to the method we are using: Markov chains and Monte Carlo simulations, the introduction will initially be divided then merged after the funda¬mentals of both have been explained

Markov chain: Given a set, let $\mathbf{X} = X_1, X_2, ...X_n$, where the vales of the element $\mathbf{X}$ are random and within the set, can be defined as a a Markov chain if the set element X_{n+1} depends solely on on X_n. Formally, when the conditional distribution of the element X_n is modelled $\mathcal{P}(X_{n+1}|X_n)$.

Monte Carlo simulation: This is a method developed by Stanislaw Ulam and John Von Newmann. As gamblers, they decided to name the method after the popular destination for gaming in Monaco. The authors described the method as akin to throwing a dice in a casino (seems to preferably be in Monaco) and evaluating the outcome. Formally, a Monte Carlo Simulation takes a random variable

of interest and runs a prediction of the expected value some specified, say n times, and assigns the estimated value after each iteration. Once the simulation reaches the specified, n time, the value is averaged to give the sim¬ulated outcome [3]. Conceptually, to implement the method, one would take the initial Markov chain, which would traverse the parameter space solely depending on the previous value previous Xn1. If there is no guiding principle which the system would fol¬low, the points would simply move about the parameter space to iinfinitum?. To mit¬igate this, one would want to construct a system where the Markov chain would: "preserve the target distribution" [2]. In simpler terms if we are able to identify the said target distribution, and then generate samples from it, there will eventually be a creation of a new set of points, say **Y**, that will be of the same distribution as the target. Integrating this idea we will get the randomly simulated events (Monte Carlo) mixed with the last item depending Markov chain. If implemented prop¬erly, this can be very powerful, as a constructed model can explore the entire set of interest. Therefore, the samples, given a long enough run time, will converge to the true expectations of the data. Unfortunately, I continue to refer to "some time" but that is the largest constraint. Currently, it is not feasible to assume near infinite computing power, and left as is, this would be a solution dependent on infinite computing power. In a perfect scenario the MCMC set up will start from its initial value, traverse the parameter space until it finds the set we are looking to find ie., the target distribu¬tion, and then explore the entire set, finally refining the set so we can have a proper estimation of the target distribution [2]. This implies that theset would be normally distributed centered around the true expectation with a standard deviation of the MCMC Standard Error:

$$\hat{f_n^{mcmc}} \sim \mathcal{N}(\mathcal{E}_\pi[f], MCMC - SE)$$

In other words, given enough time we will find that our target function f_n^{mcmc} will, by the central limit theorem, be normally distributed, with the mean being our actual expectation and standard deviation being the MCMC standard error. This standard error is computed by taking the square root of the variance and dividing by the effective sample size. This ESS is explaining how many samples given the autocorrelation between your set are effectively being used. This can

be very useful given sometimes you may run 100,000 iterations but they are all correlated so you may find yourself with effectively 10. [2]

2.2 MCMC Algorithims

One of the most prolific MCMC Algorithms is the Metropolis-Hastings Algorithm. This is named after the first authors of two papers: Metropolis et all (1953) and Hastings in (1970). Conceptually, the algorithm follows a Markov chain Monte Carlo, as: If we begin with the explained target density, I will call π, we also now need the conditional density, q, which is defined as a "proposal". This proposal is the "proposed" next step of the chain ie: x_{t+1}. With the correct proposal value, this addition can add direction to the Markov chain, assisting it from traversing the set infinitely, and instead increasing the efficiency. Using this proposal we now can set a value of which we decide to accept this proposal and move the chain to that position $x_{t+1} = proposal$, or reject and remain where we are $x_{t+1} = x_t$ [8].

$$Generate\, Y \sim q(y|x_n)$$

$$x_{t+1} = \begin{cases} Y_t\ with\ probability\ p(x_t, Y_t) \\ Y_t\ with\ probability\ p(x_t, Y_t) \end{cases}$$

$$p(x, y) = min\left\{\frac{\pi(y)}{\pi(x)} * \frac{q(x|y)}{q(y|x)}, 1\right.$$

[8] In an implemented algorithm, this function would iterate from the initial position to the final position, some specified t steps. In an ideal world we would have sufficiently sampled from the posterior distribution, and our new set of points would be similarly distributed.

2.3 Random Walk

The organic next step is to talk about the algorithm called "Random Walk" Metropolis. This was identified in the paper above by Metropolis in 1953. In this we can define our Random walk as follows:

$$q(y|x) = x + \epsilon$$

$$where\ \epsilon \sim g\ for\ some\ g\ symmetric\ about\ 0$$

$$Thus\ q(y|x) = g(\epsilon)$$

and since it is symmetric the probability of $q(x|y) = g(-\epsilon) = g(\epsilon)$ this is all to show that the equation from the above example will cancel out leaving us with a proposal value of simply[7]

$$proposal = min\left\{\frac{\pi(y)}{\pi(x)}, 1\right.$$

We then take this value and compare it to a uniformdistribution between 0 and 1 $U \sim (0, 1)$ and if this is greater, we accept the new movement else we remain in the same position [7].

3 Hamiltonian Monte Carlo

3.1 Hamiltonian Dynamics

While extremely effecting under certain conditions, the Random Walk algorithm can have serious shortfalls in high dimensional spaces. The increased magnitude of directions the chain can move in, combined with only a very small subset of properly acceptable values can lead to computing capacity issues and largely inefficient algorithms [2]. Hamiltonian Monte Carlo looks to solve this problem by integrating so-called physical properties into the algorithm. I will introduce the physical dynamics by topic, and supplement the rigorous mathematics and constraints further in the paper. Generally the largest problem one is looking to solve when making predictions is that of information. Relating back to the directionless Markov chain, with Hamiltonian Dynamics, we are looking for a method to derive a direction for the t+1 step of the chain. The first idea would be to integrate knowledge from physical systems and implement a gradient vector into the desired probability space, in order to inform the decision making process of the chain. This would look like a vector field; if we are trying to traverse around a circle, the gradient vector field at each point would be directing us around in a circle. The drawback of this, is the physical properties of a gradient function would

cause the vector to point central to the circle instead of actually traversing around, leading the chain to converge to the center and not explore the set, as a satellite in space would fall into earth if it did not have enough momentum to continue to orbit the planet.

This may not make that much sense, but the proper way to understand this would be to think of it as a satellite system. When we send a satellite into outer space, it has to have enough momentum and the correct directional pull in order to stay in orbit. Too much and it will exceed the gravitational pull and be sent out into space in the direction, too little and gravity will pull it in. That specifically is what we want to achieve on our typical set with our directions and what the Hamiltonian Equations implemented in the Markov chain Monte Carlo method look to solve [2].

We will attempt to achieve this equilibrium momentum of a satellite if you will with Hamiltonian Dynamics. To understand this we must introduce the concept of the phase space. This space is

> Conservative dynamics in physical systems requires that volumes are exactly preserved. As the system evolves, any compression or expansion in position space must be compensated with a respective expansion or compression in momentum space to ensure that the volume of any neighborhood in position-momentum phase space is unchanged. In order to mimic this behavior in our probabilistic system we need to introduce auxiliary momentum parameters, pn, to complement each dimension of our target parameter space [2]

The challenge we face is to extend an idea from Physics to a statistical model. This is done by constructing a momentum parameter which the researcher is responsible for finding. For the sake of explanation, momentum at a certain time will be denoted as p_n, meaning that for every item in our parameter space, q_n has a complementary p_n, and thus we increase the dimension now from D to 2D where D is the number of dimensions [2]. Because of this addition, all of the initial equations must be altered to account for this extra dimension.

$$\pi(q, p) = \pi(p|q)\pi(q)$$

The joint distribution we have created with momentum and position in Hamiltonian Monte Carlo is called canonical distribution [2]. The powerful part of Hamiltonian Monte Carlo, is that through the dynamics, and introduction of momentum, we can define that canonical distribution as a function of the Hamiltonian function, which given the same inputs is non stochastic [2]

$$\pi(q, p) = e^{H(q,p)}$$

Th definition lets us decompose our target joint density function into two separate functions of energy: Potential and Kinetic.

$$H(q, p) = -\ln \pi(q, p) = -\ln \pi(p|q) - \ln \pi(q) = K(p, q) + V(q)$$

[2] Now that we have translated our target densities into Kinetic (K) and Potential(V) energies, we can apply the Hamiltonian differential equations in order to integrate to a position where we can get these vectors we wanted in the ideal state.

$$\frac{\mathrm{d}q}{\mathrm{d}t} = +\frac{\partial H}{\partial p} = \frac{\partial K}{\partial p}$$

$$\frac{\mathrm{d}p}{\mathrm{d}t} = -\frac{\partial H}{\partial q} = \frac{\partial K}{\partial q} - \frac{\partial V}{\partial q}$$

An extremely important outcome of this to takeaway is that our $\frac{\partial V}{\partial q}$ is the gradient of the logarithm of the target density [2]. This equation is integral to being able to simulate Hamiltonian Monte Carlo because it gives us the ability to derive the gradients we need to keep our satellite in orbit (example) by going through the momentum we chose instead of the parameters [2] Using those gradients we can move through some t time we project back down to the typical set and then have this efficient exploration [2]

3.2 Using HMC to Create Efficient Markov Transitions

This theoretical discussion is pointless to us if we can't create an effective way to apply this in order to create a transition we want. The steps to doing this involve

- Taking our initial point in the parameter space and lifting it in a one to one transformation in the phase space [2] $p \sim \pi(p|q)$
- If we did this correctly, this point would be in the target distribution space, therefore if we were to sample the momentum directly from the conditional distribution the lift itself will again fall into the typical set on phase space. [2] $(q, p) \rightarrow \psi_t(q, p)$
- We then will project back down to the target space thus we are using the phase space to move and back to the target in order to actually explore what we want $(q, p) \rightarrow q$

This is a distinct advantage of Hamiltonian Monte Carlo because we are able to explore the target distribution or typical set much faster than say Metrpolis Hastings if parameters are chosen correctly

3.3 Efficiency

To explain the possibilities for efficiency in Hamiltonian Monte Carlo we have to get into the specific geometries of the so-called Phase Space. The idea is given the Hamiltonian Equation:

$$H^{-1}(E) = q, p|H(q, p) = E$$

[2] Now we are able to decompose the distribution equation $/pi(q, p)$ so that we are left with concentric circles whose circumference only depend on the stated energy.

$$\pi(q, p) = \pi(\theta_E|E)\pi(E)$$

This decomposition leaves the first portion of the right hand side of equation the micro canonical distribution and the right-hand side to be the marginal energy distribution [2]. The illustration would be to consider some sort of dimensions of flat circle surfaces, which then will be lifted and put back down.

3.4 Optimising Kinetic Energy

Since Kinetic Energy in the Hamiltonian Monte Carlo Model is the conditional distribution over the momentum [2] both of which we choose, this can be altered to obtain the best possible results. Kinetic Energy is the portion of the probability function that will dictate the shape of the geometry. The optimum scenario would be to construct the conditional distribution over the momentum (KE) such that we create circular sets sitting on the same plane. Picking the ideal Kinetic Energy would be computationally impossible given the infinite possibilities hence there are a few methods available.

- Euclidean-Gaussian Kinetic Energies [2]
  - $K(q,p) = \frac{1}{2}p^T \times M^-1 \times p + \ln(M) + Const$
    where M is Elucidean metrics
- Riemannian-Gaussian Kinetic Energies [2]
  - $K(q,p) = \frac{1}{2}p^T \times \Sigma^-1(q) \times p + \frac{1}{2}\ln(\Sigma^-1(q)) + Const$
- Non-Gaussian Kinetic Energies [2]
  - Any values that aren't Gaussian can be derived, though in states such as when the $\pi(E) \longrightarrow \mathcal{N}$ you'd only be able to use Gaussian for an optimal solution

3.5 Optimising Integration Time

The importance of choosing a proper value of T or integration time, is that it dictates the ability of the Hamiltonian Model to traverse the set of interest [2]. It, as in many models, is a matter of tradeoffs. If integration time is too short, then the entire set may not be properly examined, but conversely if the integration time is too long the trajectory may circulate back to the initial region. Betancourt explains the concept of Dynamic Ergodicity in his paper [1] which compares this process again to a satellite in orbit. The trajectory of the orbit is mapped and assumed that with an integration time approaching infinity, you will have explored the entire surface of interest. The premise of Dynamic Ergodicity is increasing T,

integration time, the expectation of the satellites trajectory will converge to the orbit i.e. all bases are covered [1]. While this is all very well, the most important portion is the time it takes for the values to converge.

3.6 No U-Turn Sampler

In a paper by Hoffman and Gelman in 2011: " The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo" [5] they proposed the idea of choosing the best integration time or path length for a Hamiltonian Model. This is a trade-off in which one is trying to minimize the and pick the most optimal choice. Instead of choosing, the No-U-Turn Sampler implements is a recursive algorithm in order to have an optimal trajectory. This mean that the sampler will actually end the simulation once the trajectory turns on itself [5]. The power of this being that it is not computationally extensive, and therefore can be implemented in languages like BUGS where it is more of a plug-and-chug approach.
While it is simple to understand i.e.: No-U-Turn once we start turning on ourselves we should kill the algorithm, the practice of derivation is quite rigorous. The goal of the sampler is to extend Metropolis-Hastings by removing the inherent randomness of the random walk, but doing this without having to initialize a fixed value for steps (Leapfrog Steps) [5] and thus giving the model an indication it has traversed the set enough. The first step, is for some location and momentum (p,q) we are to take the dot product between the current momentum and distance from initial point to current location, which is the derivative of the squared distance of initial position and current position over 2

$$\frac{d}{dt}\frac{(\hat{\theta}-\theta)}{2} = (\hat{\theta}-\theta) * \hat{r}$$

[5] At this point we have half of the algorithm but the last necessary piece is to integrate the concept of reversibility. The No-U-Turn Sampler does this by implementing the recursive principles that are similar to slice sampling [5].

4 HMC's Financial Applications

4.1 Systemic Risk Allocations

Hamiltonian Monte Carlo has practical applications within Finance, and more specifically in portfolio risk management. Risk allocation is an integral component of managing a profit generating portfolio as it assists in the decomposition total risk. In the paper "Systemic Risk Allocations" Takaaki Koike and Marius Hofert [6] specifically focus on the concept of Systemic Risk. Implicit to the name, this systemic risk is external to the portfolio, it is predicated on the entire financial system it operates. It is the measurement of the specific financial distress a macro-economy experiences [6]. There are different quantifications of this risk that have been proposed: conditional VaR (CoVaR) (Adrian and Brunnermeier (2016)), conditional expected shortfall (CoES) (Mainik and Schaanning (2014)) and marginal expected shortfall (MES) (Acharya et al. (2017)) [6]. such quantifications are extending the traditional Value at Risk and Expected Shortfall and adding marginal/conditional components of systemic risk. To measure this, from what is described above, one would think we could apply some MCMC method, maybe the Metropolis-Hastings algorithm in order. to solve our problem and model this systemic risk. As described, there are some shortcomings that have to be taken into account. In the typical MH algorithm, we create this α value or the acceptance probability, in random walk it would be uniform between 0,1 but nonetheless it would be an option. The problem that would be encountered when modelling the systemic risk or financial crisis events would be that the "Markov chain has serial correlation" [6] which is a detriment to the estimator as it becomes increasingly less efficient. This drawback is the antithesis to the ideal outcome of a Metropolis-Hastings algorithm as when something such as systemic risk is the function to be modelled, we end up with over-rejection as this event is a one in n (small probability) outcome event, however we want that event to be considered not rejected [6]

The power of the Hamiltonian Monte Carlo and Hamiltonian dynamics is emphasized through this problem. Using the dynamics the correlation can be mitigated and we are able to increase the probability of acceptance. Most powerfully:

"HMC candidates always belong to the crisis event by reflecting the dynamics when the chain hits the boundary of the constraints;" [6] which is the reflection property of the HMC method. Using HMC the researchers were able to model the systemic risk successfully.

For Hamiltonian Monte Carlo the tweaking of the model comes down to choice of step size and time of integration. For the application with Systemic Risk, in order to determine proper parameters and $\mathcal{T}$, they built a Monte Carlo algorithm before implementing the HMC method. This utilized inputs of this presample, the Kinetic and Potential Energy gradients as well as the target acceptance probability they indicated they would like to see. This algorithm then using the Leapfrog Integration to derive alphas and new gradients give the time step t, gave the correct outputs for what we want. With the aforementioned parameters they then ran the Hamiltonian Monte Carlo Algorithm, giving the output $X_1, ...X_n$ of the Markov chain. This research was extremely successful in the application as it used two different runs of $(\epsilon, T) = 1 \sim (0.210, 12)\ 2 \sim (0.095, 13)$ which yielded acceptance probabilities asymptotically approaching 1 [6].

Using this successful study, they implemented a proper empirical study on insurance liability claims, and aimed to use this information. This study proved the positive use case of HMC by deriving a smaller standard error and emphasizing that using the methods of simple MC you may overstate by not properly weighting the big losses with how infrequent they appear. However, the drawback is that target distributions that have fat tails can be extremely computationally intensive if the ideal step size, ϵ, is small and the integration time, T [6]

4.2 Stochastic Volatility Models

In a different study, Hamiltonian Monte Carlo methods were used in conjunction with financial time series data to model stochastic volatility. To address some of the issues stated above (fat tails, asymmetry) the researchers proposed different ideas of distributions to the generally accepted normal. [4]. To apply the stochastic volatility theory to practice, their focus utilized foreign exchange, "Forex" data comparing the British Pound to the United States Dollar (£,USD) and Euro to USD (EUR/USD) as well as data provided from the Brazilian stock exchange in

Sao Paulo. [4] The success of this study was in creating parameter estimates that had beneficial information criteria based upon the Widely Applicable Information criterion (WIAC) and Leave-One-Out Cross Validation (LOO) [4].

4.3 Literature Summary

The selected publications are included, first, to emphasize the power of Hamiltonian Monte Carlo in practice, and second to encourage the propulsion of quantitative finance research using HMC methods. There can be significant advances in Financial Mathematics and Financial Time-Series Modelling if more Bayesian statistical methods are implemented. There are extremely successful Stochastic Volatility Models for many years, and there is room for continued optimisation using Hamiltonian Monte Carlo to increase the speed and precision of deriving information baring estimators.

5 Implementation

In order to give a tangible explanation of Markov chain Monte Carlo methods, pseudo code the for algorithms has been provided below. The appendix contains R code that can be run to see example results.This is in order to emphasize the sophistication yet simplicity of working with Hamiltonian Monte Carlo and to demonstrate how flexible it can be in application.

5.1 Algorithms

Algorithm 1 Markov chain Monte Carlo (Simple)

Require: Prior Distribution ex: Exponential Number of Iterations Starting Value, Proposal SD

$x =$ Vector of 0s with length Iteration

$X[1] =$ Starting Value

for do $2 : Iterations$

$current_x = x[i-1]$

$proposed_x \sim \mathcal{N}(current_x, proposal_s d)$ ▷ This is a comment

$\alpha = priorDist(proposed_x)/prior_dist(current_x)$

if $\alpha >$ Random $\in$ (0,1) **then**

$X_i = proposed_x$

else

$X[i] = current_x$

end ifreturn $X_1, .., X_{Iterations}$

Algorithm 2 Hamiltonian Monte Carlo

Require: Prior Distribution ex: Exponential, Potential Energy, Gradient of The Potential Energy, Step Size, Integration Length, Current Position

$q = current_q$

$p = \sim \mathcal{N}(length(q), 0, 1)$

$p = current_p$

$p = p - \epsilon * \frac{PEGradient}{2}$

for do $1 : Integration\ Steps$

$q = q + \epsilon * p$

if $if(i! = IntegrationSteps)$ **then**

$p = p - \epsilon * PEGradient$

$p = -p$

$current_U = U(current_q)$

$current_K = sum(current_p^2)/2$

$proposed_U = U(q)$

$proposed_K = \frac{\sum(p^2)}{2}$

if $(0,1) < Current_U - Proposed_U + Current_K - Proposed_K$ **then**

$return(q)$

else

$return(current_q)$

return $X_1, ..., X_{iterations}$

end if

end if

6 Empirical Example With Data

6.1 Introduction

To show the difference between MCMC Random Walk and Hamiltonian Monte Carlo, I will illustrate an example using R to show. I will first run a linear regression model in order to compare with frequentist estimates, followed by a Markov chain Monte Carlo Random Walk Metropolis and Hamiltonian Monte Carlo. I will compare point estimates, as well as credible intervals with speed. Although these datasets are not going to be as cumbersome possible others in a speed comparison, they should nonetheless be worthy examples. To give a practical example of Hamiltonian Monte Carlo, I will implement the method in R. This example will compare a frequentist method of Linear Regression with a Hamiltonian Monte Carlo Simulation. The goal will be to sample data from a normally distributed posterior and return a set of new points from the Markov chain that will also be normally distributed. I will overlay the Linear Regression expected mean response, with our Hamiltonian Monte Carlo simulation for comparison.

6.2 Data Set

For simplicity of explanation I will be using the R-Base data set "IRIS" that comes with the R language [9]. I attempt to predict the Length of the Petal (Continuous Variable) with the Species (factor with three inputs).

6.3 Methods

For the simulation I will use the HMC learn [10] in R [9] and the lm linear regression function in r. To set up the linear regression I will need to build the linear model:

$$y_i = \mathbf{x}_i^T \beta + \epsilon_i$$

where $\mathbf{X}\ and\ \beta$ are vectors of the predictor variables and corresponding coefficients. For this example, I will assume the assumptions of linear regression that the error terms e_i are normally distributed with $\mu = 0$ and constant variance. The

subsequent step will be to derive the log likelihood function for my linear regression function which are:

$$\ln f(\mathbf{y}|\beta, \sigma^{\mathbf{2}}) \propto -n\ln(\sigma) - \frac{1}{2\sigma^2}(\mathbf{y} - \beta\mathbf{X})^T(\mathbf{y} - \beta\mathbf{X})$$

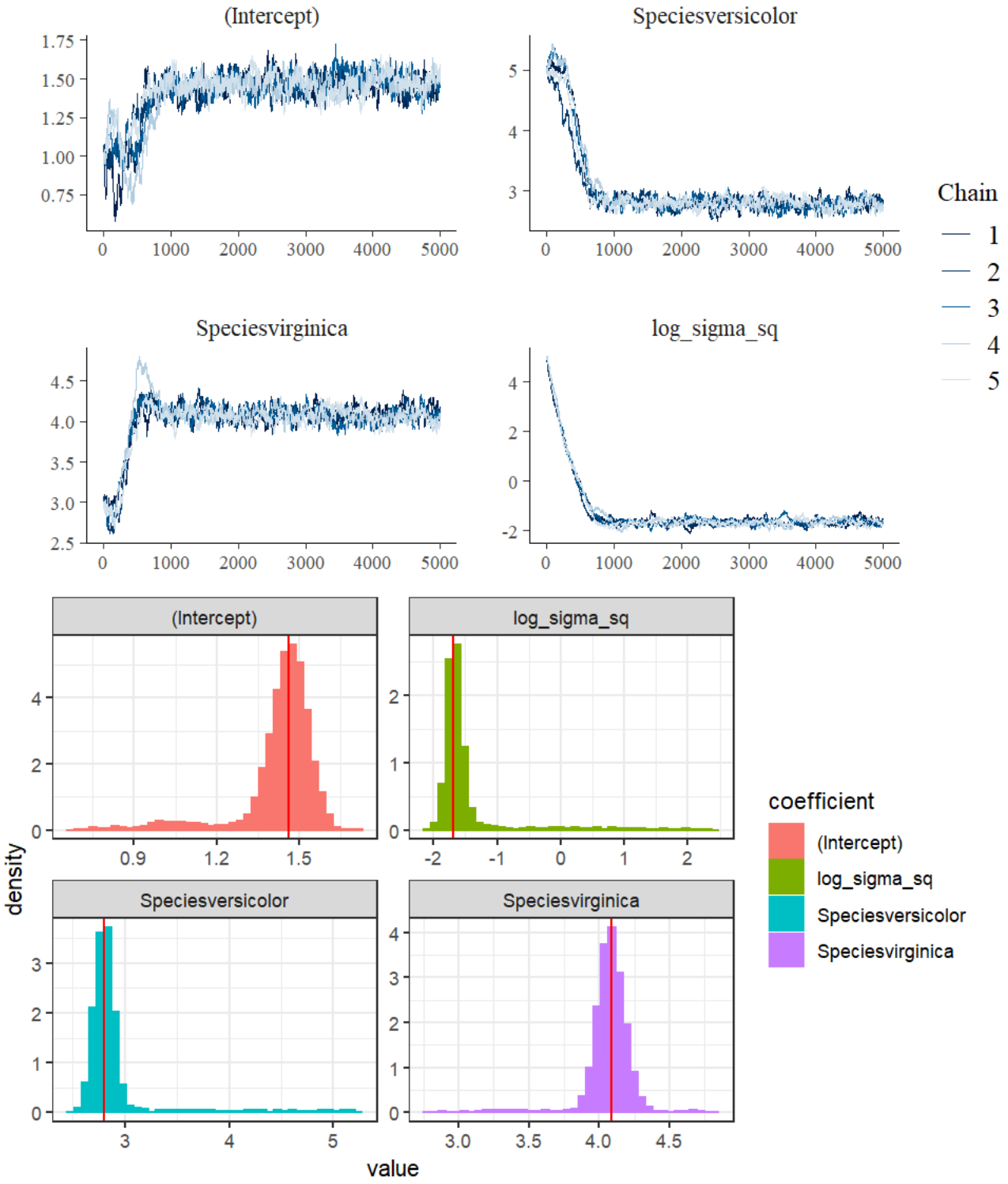
[11]. As standard for Hamiltonian Monte Carlo in this scenario I will be using priors for the vector of coefficients β as a Multivariate Normal, as well as an inverse gamma for the variance of the residuals. [11]. For Hamiltonian Monte Carlo I need to derive the log posterior with which I can model with, simply the log prior + the log likelihood. Skipping some computation for the sake of simplicity, one is left with the following relationship, relating the priors and likelihoods to the Hamiltonian Monte Carlo and the Hamiltonian Equations:

$$H(\theta, \mathbf{p}) \propto logf(\beta, \gamma|y, X, \sigma_\beta^2, a, b) + \frac{1}{2}p^T M^{-1} p.$$

[11] After deriving these I then need the Gradients of the functions. As these are very computationally heavy using the differential equations explained above, I will use the HMC Learn package [10] in order to derive these using the computer. Once I do this we can use the linear regression and the HMC MCMC to compare. The code used to implement these methods are included in the appendix.

6.4 Results

Using the HMC Methods we were able to achieve convergent Markov chains as well as effectively sample from the posterior distribution. Compared with the frequentist estimates (red line) we were able to create a distribution centered around the same expected mean response from linear regression. Though a simple example with a cleaned data set, the framework is laid for further implementation. For this model the chosen parameters were 5000 iterations, initial values (1,5,3,5), $\epsilon = 0.002$, leapfrog step length = 0.2 and 5 chains.


(Intercept)
Speciesversicolor
Speciesvirginica
log_sigma_sq
Chain
1
2
3
4
5
density
value
coefficient
(Intercept)
log_sigma_sq
Speciesversicolor
Speciesvirginica

7 Final Thoughts

I began this paper with a survey of Monte Carlo, Markov chains, Markov chain Monte Carlo, Metropolis-Hastings. I then moved into the deep and powerful algorithm of the Hamiltonian Monte Carlo simulation. Then moved into a survey of the current literature related to finance and then a simple example of how this can be empirically implemented. The aim of this paper was to give an overview of the algorithm in theory and practice. I think there is great potential within finance, particularly related to modelling stochastic volatility and portfolio risk. Writing this paper has motivated me to seriously pursue further research within this field. I hope to contribute to the current academic literature by using Hamiltonian Monte Carlo methods to model portfolio risk specifically within the Delta-Vega Hedging theory in Financial Mathematics.

References

[1] Michael Betancourt, *Identifying the optimal integration time in hamiltonian monte carlo*, 2016.

[2] Michael Betancourts, *A conceptual introduction to hamiltonian monte carlo*, 2018.

[3] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng, *Handbook of markov chain monte carlo*, Taylor and Francis, 2011.

[4] David S. Dias and Ricardo S. Ehlers, *Stochastic volatily models using hamiltonian monte carlo methods and stan*, 2017.

[5] Matthew D. Hoffman and Andrew Gelman, *The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo*, 2011.

[6] Takaaki Koike and Marius Hofert, *Markov chain monte carlo methods for estimating systemic risk allocations*, 2020.

[7] Roger D. Peng, *Advanced statistical computing*, Johns Hopkins Advanced Statistical Computing Course, 2021.

[8] Christian P. Robert, *The metropolis-hastings algorithm*, 2016.

[9] R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2021.

[10] Samuel Thomas, *hmclearn: Fit statistical models using hamiltonian monte carlo*, 2020. R package version 0.0.5.

[11] Samuel Thomas and Wanzhu Tu, *Learning hamiltonian monte carlo in r*, 2020.

8 Appendix

8.1 Example of implementation of a simple MCMC

Establish an Exponential Prior

```
prior_dist = function(x){
  if(x<0){
    return(0)}
  else {
    return( exp(-x))
  }
}
```

```
Create the MCMC function
MCMC_Function = function(iterations, starting_val, proposal_sd){
  x = rep(0,iterations)
  x[1] = starting_val
  for(i in 2:iterations){
    currentx = x[i-1]
    proposedx = rnorm(1, mean = currentx, sd = proposal_sd)
    alpha = prior_dist(proposedx)/prior_dist(currentx)
    if(runif(1) < alpha){
      x[i] = proposedx
    }else{
      x[i] = currentx
    }
  }
  return(x)
}

#Now run some chains with differeing parameters
run_chains = function(iter,init,sd){
chain1 = MCMC_Function(iter,init,sd)
chain2 = MCMC_Function(iter,init,sd)
chain3 = MCMC_Function(iter,init,sd)

plot(chain1,type="l")
lines(chain2,col=2)
lines(chain3,col=3)   }
run_chains(10000,3,1)
```

[11]

8.2 Creating an HMC Implementation from Scratch

[3]

```

HMC = function (U, grad_U, epsilon, L, current_q){
  q = current_q
  p = rnorm(length(q),0,1)  # independent standard normal
    variates
  current_p = p
```

```
  # Make a half step for momentum at the beginning
  p = p - epsilon * grad_U(q) / 2
  # Alternate full steps for position and momentum
  for (i in 1:L){
    # Make a full step for the position
    q = q + epsilon * p
    # Make a full step for the momentum, except at end of
    trajectory
    if (i!=L) p = p - epsilon * grad_U(q)
  }
  # Make a half step for momentum at the end.
  p = p - epsilon * grad_U(q) / 2

  # Negate momentum at end of trajectory to make the proposal
    symmetric
  p = -p
  # Evaluate potential and kinetic energies at start and end of
    trajectory
  current_U = U(current_q)
  current_K = sum(current_p^2) / 2
  proposed_U = U(q)
  proposed_K = sum(p^2) / 2
  # Accept or reject the state at end of trajectory, returning
    either
  # the position at the end of the trajectory or the initial
    position
  if (runif(1) < exp(current_U-proposed_U+current_K-proposed_K))
  {
    return (q)  # accept
  }
  else
  {
    return (current_q)  # reject
  }
}
```

The equations for the Logarithm of the Posterior and the Gradient of the Log Posterior:

```
Log_Posterior = function (theta, y, X, sig2beta = 1000) {
```

```
    k <- length(theta)
    beta_param <- as.numeric(theta)
    onev <- rep(1, length(y))
    ll_bin <- t(beta_param) %*% t(X) %*% (y - 1) - t(onev) %*%
        log(1 + exp(-X %*% beta_param))
    result <- ll_bin - 1/2 * t(beta_param) %*% beta_param/
    sig2beta
    return(result)
}
Gradient_Log_Posterior = function (theta, y, X, sig2beta = 1000)
    {
    n <- length(y)
    k <- length(theta)
    beta_param <- as.numeric(theta)
    result <- t(X) %*% (y - 1 + exp(-X %*% beta_param)/(1 + exp
    (-X %*%
        beta_param))) - beta_param/sig2beta
    return(result)
}
```

[10]

8.3 Implementation Code

```
library(hmclearn)

#Set Initial Values and Matrix Models
y = iris$Petal.Length
X = model.matrix(Petal.Length~Species,
                 data = iris)
#Initialize the Parameters
iters = 5000
inits = c(1,5,3,5)
epsil =  c(rep(2e-2, 3), 2e-2)
#build the Hamiltonian Monte Carlo Model Using HMCLearn [1.1]
iris_hmc <- hmc(iters,
                theta.init = inits,
                epsilon = epsil,
                L = 0.2,
                logPOSTERIOR = linear_posterior ,
```

```
                glogPOSTERIOR = g_linear_posterior,
                varnames = c(colnames(X), "log_sigma_sq"),
                param = list(y = y, X = X), chains = 5,
                parallel = FALSE)
#Show the MCMC TRace
mcmc_trace(iris_hmc)
#Initialize the Model to compare with freqentist
comparison_model = lm(Petal.Length~Species,
                      data = iris)
freq.param <- c(coef(comparison_model), 2*log(sigma(comparison_
    model)))
#Plot with a burn in
diagplots(iris_hmc, burnin=200, comparison.theta=freq.param)
```