

# Testing Magnetic Hamiltonian Monte Carlo on the Sine Gordon Model

A thesis submitted in partial fulfillment of the requirement  
for the degree of Bachelor of Science in  
Physics from the College of William and Mary in Virginia,


by

Preston C. Mulford



---

Advisor: Prof. Konstantinos Orginos



---

Todd Averett

Prof. Todd Averett

Williamsburg, Virginia  
May 7th, 2021

# Contents

Acknowledgments	iii
List of Figures	iv
List of Tables	v
Abstract	v
<b>1 Introduction</b>	<b>1</b>
<b>2 Theory</b>	<b>2</b>
2.1 Monte Carlo . . . . .	2
2.1.1 Markov Chain Monte Carlo . . . . .	2
2.1.2 Metropolis-Hasting Accept/Reject Step . . . . .	3
2.2 Hamiltonian Monte Carlo . . . . .	4
2.2.1 Hamiltonian Monte Carlo Leapfrog Integrator . . . . .	5
2.3 Magnetic Hamiltonian Monte Carlo . . . . .	6
2.3.1 MHMC Leapfrog Integrator . . . . .	6
2.4 Quantum Harmonic Oscillator . . . . .	7
2.5 sine-Gordon . . . . .	9
2.6 Autocorrelation . . . . .	10

<b>3 Experiments</b>	<b>12</b>
3.1 $\epsilon^2$ Test . . . . .	12
3.2 Reversibility Tests . . . . .	13
3.3 QM Tests . . . . .	14
3.4 sine-Gordon Tests . . . . .	17
3.4.1 4x4 Lattice . . . . .	18
3.4.2 16x16 Lattice . . . . .	22
3.4.3 32x32 Lattice . . . . .	23
<b>4 Conclusion/Outlook</b>	<b>26</b>
<b>A Example Leapfrog Integrators</b>	<b>27</b>
A.1 HMC Integrator . . . . .	27
A.2 MHMC One-Dimensional Integrator . . . . .	27
A.3 MHMC Two-Dimensional Integrator . . . . .	28
<b>B Autocorrelation</b>	<b>29</b>
B.1 Example Autocorrelation Function . . . . .	29

# Acknowledgments

I first have to give a massive thank you to Dr. Orginos for his constant support and patience through this process. This has been quite the struggle to say the least and I had no idea what this was going to turn into when we first started this research. I've learned more than I ever thought I would and still understand very little. This stuff is hard I would also like to say thank you to my family and friends who have supported me in this process. There have definitely been quite of few moments of me wandering through the apartment driving all roommates crazy as I tried to fix whatever problems I was having. In the end, so long and thanks for all the fish.

# List of Figures

3.1	QM $\epsilon^2$ Test . . . . .	13
3.2	SG $\epsilon^2$ Test . . . . .	13
3.3	QM Autocorrelation . . . . .	15
3.4	QM $X^2$ Results . . . . .	16
3.5	QM Results . . . . .	17
3.6	SG HMC Small Lattice . . . . .	18
3.7	SG MHMC $G_1$ 4x4 Lattice . . . . .	19
3.8	SG MHMC $G_2$ 4x4 Lattice . . . . .	20
3.9	SG MHMC $G_3$ 4x4 Lattice . . . . .	22
3.10	SG HMC 16x16 Lattice Tests . . . . .	22
3.11	SG MHMC $G_3$ 16x16 Lattice Tests . . . . .	23
3.12	SG HMC 32x32 Lattice Tests . . . . .	24
3.13	SG MHMC $G_3$ 32x32 Lattice Tests . . . . .	25

# List of Tables

3.1	QM Reversibility Tests . . . . .	14
3.2	SG Reversibility Tests . . . . .	14

## **Abstract**

The goal of this paper is to explore the efficacy of Magnetic Hamiltonian Monte Carlo (MHMC) on a Quantum Harmonic Oscillator and sine-Gordon model. The traditional method to solve Lattice Quantum Chromodynamics (QCD), and other high-dimensional integrals, Hamiltonian Monte Carlo (HMC), gets stuck when exploring these distribution and fails to sample from the entire distribution. MHMC, when initially proposed as an alternative to HMC, showed an ability to sample from all regions of an action rather than getting bogged down in singular results as HMC tends to do. In this paper we apply MHMC to a simple one-dimension model, Quantum Harmonic Oscillator, and a more complicated two-dimensional sine-Gordon model and compare its results with HMC. In our tests, on both the sine-Gordon and Quantum Mechanical models, MHMC failed to show any improvement on HMC's sampling and often time proved to be less efficient than HMC when performing these calculations.

# Chapter 1

## Introduction

In Lattice Quantum Chromodynamics, calculating complex models requires intense, multidimensional integrals that prove difficult to solve. The canonical method for solving these integrals is known as Hamiltonian Monte Carlo (HMC). HMC, in theory, samples from and explores the entire distribution. In practice, HMC often fails to give a full description of the distribution and gets stuck in specific regions. This is due to computational limits. Given infinite computing power and infinite time, HMC will sample from the entire distribution. In 2017, a new non-canonical algorithm known as Magnetic Hamiltonian Monte Carlo (MHMC) was proposed as an alternative to HMC. [2] In this paper MHMC showed itself to be more efficient when sampling from a weighted Gaussian mixture and the FitzHugh-Nagumo model. In this we explain both the HMC and MHMC algorithms before applying them to a simple one-dimensional Quantum Anharmonic Oscillator and the sine-Gordon model. To this end, we replicated MHMC algorithm and corresponding leapfrog integrator. We then compared MHMC to HMC for the one-dimensional case as well as the two-dimensional case using lattices of different sizes. In particular we are interesting in comparing the autocorrelation times between the two algorithms in an attempt to see differences in efficiency.



# Chapter 2

## Theory

### 2.1 Monte Carlo

Monte Carlo is a method through which a probability distribution is randomly sampled to approximate a desired result. This algorithm category allows for complicated path integrals to be approximated numerically where analytic solutions fail. Essentially, Monte Carlo proposes a random point and decides the likelihood of that point being a part of the distribution. By sampling a large number of points, the distribution, in our case a path integral, can be approximated. The more points sample, the more accurate the description of the path integral.

#### 2.1.1 Markov Chain Monte Carlo

Markov Chain Monte Carlo is a method through which the sampling is no longer completely random. As outlined in "A Simple Introduction to Markov Chain Monte Carlo sampling", rather than sampling a completely new position each time, Markov Chain Monte Carlo proposes a new position based off of the previous position.[4] The only possible new positions are thus the neighboring positions rather than any position within the bounds of the experiment. This trail of samples, i.e. neighboring

point, are thus described as a Markov Chain. It is important to note, that the Markov Chain does not discriminate on direction. Instead each neighbor is equally probable, resulting in a random walk around the distribution. What Monte Carlo does with the Markov chains is determine whether the proposed point is part of the distribution or not. If the point is part of the distribution, it is accepted, and the random walking is repeated. However, if rejected, the Markov Chain process is repeated until a neighbor is proposed that is accepted. For the purposes of this experiment, we use a Metropolis-Hastings accept reject to discriminate between points.

### 2.1.2 Metropolis-Hasting Accept/Reject Step

The accept/reject step is the same for both HMC and MHMC. As described in Lattice QCD for Novices, if the change in the Hamiltonian, i.e. the energy, is negative, then the new point is accepted.[1] If the change in energy is non-negative, then we raise  $e$  to the negative change in the Hamiltonian ( $e^{-dH}$ ) and compare this term to a random Gaussian number in the range of 0 to 1. If  $e^{-dH}$  is greater than the random Gaussian, the new value of  $x$  is accepted. However, if the opposite is true, the initial  $x$  is refused with a new random disturbance. Therefore, the larger the change in energy is, the less likely it is to be accepted in this step. The only variation between HMC and MHMC during this process is how MHMC handles the G-matrix, which is unique to MHMC, and the momenta. HMC only returns either  $x_{new}$  or  $x_{initial}$  through this process. MHMC accept/reject returns both the position and the momenta for all cases. Additionally, in cases where the  $x_{new}$  is rejected, the sign of the G matrix and momenta are flipped before the process repeats itself again.

## 2.2 Hamiltonian Monte Carlo

The Hamiltonian Monte Carlo algorithm (HMC) is a variation of Markov Chain Monte Carlo where the changes in energy, as described by Hamiltonian dynamics, are used to define the accept/reject criteria. Rather than just randomly sampling position, HMC uses a random momentum to disturb the current configuration before performing a Metropolis-Hasting accept/reject step. This momentum term is independent of position, as it is randomly generated, but of equal size to the position term. The Hamiltonian is the sum of the kinetic and potential energies of the particle in question. It follows, that by changing the momentum, we can update the Hamiltonian to represent this change. In simple cases, like the one-dimensional Quantum Harmonic Oscillator, this is easily done analytically. However, as the dimensions of the integral increase, we can no longer solve it analytically and instead must numerically evaluate this transition using leapfrog integration. The HMC algorithm in discrete terms is as follows.

---

**Algorithm 1:** Hamiltonian Monte Carlo

---

```
Input: H, L,  $\epsilon$ 
Initialization( $x_0, p_0$ )
for  $n = 1, \dots, N$  do
    Refresh  $p_{i-1} = N(0, I)$ 
    Calculate( $x_n, p_n$ ) = LF(H, L,  $\epsilon$ ,  $x_{n-1}, p_{n-1}$ )
    if  $Unif([0, 1]) < \min(1, \exp(H(x_{n-1}, p_{n-1}) - H(x_n, p_n)))$  then
        | Return ( $x_n, p_n$ )
    else
        | Return ( $x_{n-1}, p_{n-1}$ )
end
```

---

In this algorithm, H is the Hamiltonian, L is the lattice and  $\epsilon$  is the step size the leapfrog integrator uses.

### 2.2.1 Hamiltonian Monte Carlo Leapfrog Integrator

In general leapfrog integration is a method to numerically integrate differential equations. For HMC the position  $X$  is a function of  $P$  whereas  $P$  is a function of  $X$ . The leapfrog integrator takes advantage of this relationship by leapfrogging back and forth between updating  $x$  and  $p$  over discrete number of steps. Given a randomized  $p$ , the algorithm first updates this  $p$  using a half step of the force according to  $x$ . Then, it updates  $x$  with the change in  $p$  using a full step. The process is repeated with a full step update of  $p$  this time. After a number of steps have been completed  $p$  is finally updated with a half step update which ensures an integer number of steps have taken place. The output of the leapfrog algorithm is a new momentum and position coordinate from which we can recalculate the Hamiltonian in order to complete the Metropolis-Hastings accept/reject step. The discrete description of this algorithm is as follows:

---

**Algorithm 2:** Hamiltonian Monte Carlo Leapfrog Integrator

---

**Input:**  $x_0, p_0 \in$   
 $p_i = p_0 + \frac{1}{2}F(x_0)\epsilon$   
**for**  $n = 1, \dots, \frac{1}{\epsilon}$  **do**  
     $x_n = x_{n-1} + p\epsilon$   
     $p_n = p_{n-1} + F(x_n)\epsilon$   
**end**  
 $x_n = x_{n-1} + p\epsilon$   
 $p_n = p_{n-1} + \frac{1}{2}F(x_n)\epsilon$   
Return  $(p_n, x_n)$

---

In this description,  $F(x)$  is the force calculated from the current  $x$  and  $\epsilon$  is the step size. Equations 2 and 3 are repeated  $\frac{1}{\epsilon}$  times in order to complete a full integration. The smaller  $\epsilon$  is, the more intermediate steps are taken. A python implementation of this algorithm is in A.1.

## 2.3 Magnetic Hamiltonian Monte Carlo

The overall MHMC Algorithm follows the same structure of HMC but adds an additional term,  $G$ , into the mix.  $G$  is an invertible antisymmetric matrix of size  $L^2$ .  $G$  is a so called magnetic field term which introduces a curl into the Hamiltonian dynamics. The idea being that this additional term will push any sampling algorithm over any topographical features that otherwise would result in the algorithm getting stuck. However, there is not clear ideal  $G$  term. Instead, the only constraints are that it is invertible and antisymmetric. While this does give great leeway when it comes to MHMC implementation, it also means success or failure of the algorithm could be dependent on an inefficient  $G$  matrix.

Like HMC, MHMC uses a Metropolis-Hastings accept reject step. However, the difference, outside of the leapfrog integrator, is that acceptance flips the signs of both  $G$  and  $p$  when returning those values. Algorithm 3 details MHMC in greater detail.

---

**Algorithm 3:** Magnetic Hamiltonian Monte Carlo

---

```
Input:  $H, G, L, \epsilon$   
Initialization( $x_0, p_0$ )  
for  $n = 1, \dots, N$  do  
    Refresh  $p_{i-1} = N(0, I)$   
    Calculate( $x_n, p_n$ ) = LF( $H, G, L, \epsilon, x_{n-1}, p_{n-1}, G_{n-1}$ )  
    Flip momentum ( $x_n, p_n$ )  $\rightarrow$  ( $x_n, -p_n$ )  
    Set  $G_n \leftarrow -G_{n-1}$   
end  
if  $Unif([0, 1]) < \min(1, \exp(H(x_{n-1}, p_{n-1}) - H(x_n, p_n)))$  then  
    | Return ( $x_n, p_n, G_n$ )  
else  
    | Return ( $x_{n-1}, p_{n-1}, G_{n-1}$ )
```

---

### 2.3.1 MHMC Leapfrog Integrator

The MHMC leapfrog integrator [2] follows much the same form as the HMC leapfrog integrator. Like HMC LF, MHMC LF is book ended by half step updates

of the momentum. These momentum updates are the exact same as the canonical LF. However, in the internal full step updates, position and momenta updates are different due to the addition of the G matrix. The position is updated by multiplying the exponential of G minus an identity matrix equal in size to G by the momentum. This result is the multiplied by the inverse of G before being added to the current x value. Additionally, momentum is multiplied by the G exponential and then updated again using the canonical momentum update only this time in a full step. Like the canonical example, it finishes with a half step update of momentum. The discrete description of the MHMC LF is as follows:

---

**Algorithm 4:** Magnetic Hamiltonian Monte Carlo Leapfrog Integrator

---

**Input:**  $x_0, p_0, G, \epsilon$   
 $p_n = p_0 + \frac{1}{2}F(x_0)\epsilon$   
**for**  $n = 1, \dots, \frac{1}{\epsilon}$  **do**  
     $x_n = x_{n-1} + G^{-1} [(e^{G\epsilon} - I)(p_i)]$   
     $p_n = e^{G\epsilon} p_{i-1}$   
     $p_n = p_{n-1} + F(x_n)\epsilon$   
**end**  
 $x_n = x_{n-1} + G^{-1} [(e^{G\epsilon} - I)(p_i)]$   
 $p_n = e^{G\epsilon} p_{i-1}$   
 $p_n = p_{n-1} + \frac{1}{2}F(x_n)\epsilon$   
Return  $(p_n, x_n)$

---

Of note, if G were set to 0, then MHMC LF would be the exact same as HMC LF. An implementation of this algorithm for both the one dimensional and two dimensional case is in A.2 and A.3.

## 2.4 Quantum Harmonic Oscillator

A simple model where HMC can be seen to get stuck is the Quantum Harmonic Oscillator. This model is relatively simple to implement and also easy to solve analytically rather than through Monte Carlo. For that reason, it is the perfect first test to study the efficacy of MHMC. Not only can we see clear differences in the two algorithms, but we can also check the correctness for each algorithm analytically. The

Hamiltonian of the An-harmonic Oscillator is that of the Quantum Harmonic Oscillator with an additional quartic term  $gx^4$  where  $g$  is a constant. The total Hamiltonian of the QM model is thus

$$H = \frac{p^2}{2m} + Ux + gx^4 \quad (2.1)$$

Where  $p$  is the momentum,  $\omega$  is the frequency,  $m$  is the mass and  $x$  is the position. In order to calculate the QM Action, we first have to do a Legendre transformation of the Hamiltonian. The resultant Lagrangian,

$$L = \frac{p(t)^2}{2} + U(x(t)) \quad (2.2)$$

, can then easily be plugged in and integrated over time to calculate the QM action. The action consequently, is

$$S = \int_{t_1}^{t_2} L dt = \int_{t_1}^{t_2} \frac{p(t)^2}{2} + U(x(t)) dt \quad (2.3)$$

However, in order to use HMC and MHMC to explore this model, we need a discrete form of the force that can be used by the leapfrog algorithms. The force is the derivative of the action with respect to  $x$ . Additionally, the Hamiltonian can also be described as the a the combination of the kinetic energy and the action. The easiest way to do this is to take a single step of the integral and describe its effect on the Lagrangian Rather than going from  $t_1$  to  $t_2$  we can consider the integral as a series of small integrals from  $t_i$  to  $t_{i+1}$ . At that point, we can approximate this individual step as

$$\int_{t_i}^{t_{i+1}} L dt = \frac{m}{2} x_{i+1} - x_i^2 + \frac{1}{2} V(x_{i+1}) + V(x_i) \quad (2.4)$$

With this approximation in hand, we then describe the entire action as a sum of discrete steps ranging from 0 to  $N-1$ . This approximation, given the QM model, looks as follows:

$$S(x) = \sum \left( \left( \frac{1}{2} \omega^2 + 1 \right) x_i^2 + \frac{1}{24} g x_i^4 - x_{i+1} * x_i \right) \quad (2.5)$$

The force is then just the derivative of the potential as defined by the action in terms of  $x$ , or

$$F = \frac{\delta U}{\delta x} = x_{i+1} + x_{i-1} - \omega^2 x + 2x - \frac{1}{6}gx^3 \quad (2.6)$$

We use this derivative to make small changes to  $p$  as outlined in the section 2.2.1 and 2.3.1.

## 2.5 sine-Gordon

The sine-Gordon (SG) model is a two dimensional model characterized by a nonlinear periodic term that makes it difficult to deal with using HMC. While it does not stuck like QM, its somewhat longer autocorrelation time makes it a perfect candidate to compare MHMC and HMC. If MHMC is more efficient than HMC we should see a clear difference in the in their autocorrelation times.

The potential energy of the can be described as the a quadratic of the variable  $x$ , in this case position, over both possible indices.

$$V_{SG} = \frac{1}{T} \left[ \sum_{i,j} |x_i - x_j|^2 - \sum_i \cos(x_i) \right] \quad (2.7)$$

If we considered the action in a similar way to the QM, then the action is that potential's quadratic expanded to take into account the neighbors for the interior terms. Consequently the action samples from not just the current position, but in the general region. The action, through this expansion is

$$S_{SG} = \frac{1}{T} \left[ \sum 2x_{(i,j)}^2 - x_{(i+1,j)}x_{(i,j)} - x_{(i,j+1)}x_{(i,j)} - \sum \cos(x_{(i,j)}) \right] \quad (2.8)$$

It follows that the force would be

$$F = \frac{1}{T} \left[ -4x_{(i,j)} + x_{(i+1,j)} + x_{(i-1,j)} + x_{(i,j+1)} + x_{(i,j-1)} - \sin(x_{(i,j)}) \right] \quad (2.9)$$



due the force as the action's derivative. Of course, instead of just sampling in the +i or +j direction, the force takes into account all possible neighbors which is seen in the above equation. The Hamiltonian is the sum of the kinetic energy and the action. The SG Hamiltonian is

$$H_{SG} = \sum \frac{p^2}{2} + \frac{1}{T} \left[ \sum 2x_{(i,j)}^2 - x_{(i+1,j)}x_{(i,j)} - x_{(i,j+1)}x_{(i,j)} - \sum \cos(x_{(i,j)}) \right] \quad (2.10)$$

## 2.6 Autocorrelation

Autocorrelation refers to the correlation of a current series with a lagged, or delayed version of that series. In other words, it shows how similar a given results and lagged version of that result is. For example an autocorrelation of 1 means that there is a perfect positive correlation. A positive autocorrelation in general means that an increase in the current series leads to an increase in the lagged series. On the flip side, negative autocorrelation means that an increase in the current time series leads to a decrease in the lagged series. Another way to think about this is positive autocorrelation means the two series mirror each others movement whereas a negative autocorrelation means the two series diverge from each other. An autocorrelation of 0, means that there is no relationship between the current series and the time lagged series. Therefore an increase in the current series does not correlate to anything happening to the time lagged series

For our research, we are interested in seeing how quickly the autocorrelation approaches 0 for both models and algorithms. The autocorrelation time roughly equivalent to the MCMC convergence time. MCMC convergence occurs "when the generated Markov chain converges in distribution to the posterior distribution of interest." [3] Unfortunately calculating MCMC convergence is extremely difficult which is why we use autocorrelation time as an approximation. Algorithms that create large

autocorrelation values take longer, and more iterations, to properly explore the entire distribution. Therefore, looking at the changing autocorrelation values gives us an approximate look at each algorithms efficiency when traversing different distributions. The quicker the autocorrelation gets to zero, the faster the algorithm is.

The formula for autocorrelation itself is well documented and is

$$A = \frac{\sum_{i=T+1}^{n-T} (x_{i-T} - \bar{x})(x_i - \bar{x})}{\sum_{i=1}^n (x_i - \bar{x})^2} \quad (2.11)$$

where T equals the lag time, n = number of observations in the series,  $x_i$  is the value of x at position i, and  $\bar{x}$  is the mean of the entire series. If we then graph this function over the all our results, we will be able to understand how quickly our MCMC algorithm's convergence time. The standard error for autocorrelation is

$$SE = \sqrt{\frac{1 + 2 \sum_{m=1}^{T-1} A_m^2}{n}} \quad (2.12)$$

where T is the lag,  $A_m$  is the autocorrelation of lag m and n is the number of observations. There is an example of an autocorrelation implementation in Python in B.1.

# Chapter 3

## Experiments

For the purposes of this research, we constructed lattices using python that we then could apply both MHMC and HMC and compare. That entailed one-dimensional lattice for the QM model and a two-dimensional lattice for SG. In order to test the validity of the both models and both algorithms we conducted both reversibility tests and  $\epsilon^2$  tests. After proving the validity of our implementation of QM, SG, HMC, and MHMC, we move on to comparing HMC and MHMC across both models. For these tests, we are particularly interested in the autocorrelation for both the one-dimensional case and the two-dimensional case. Additionally, for QM, we can plot the results of each loop through HMC and MHMC and visually see where these algorithms get stuck. Finally, we also want to compare the

### 3.1 $\epsilon^2$ Test

The epsilon squared test checks the correctness of the leapfrog integrator. Instead of using a single value for epsilon, the epsilon squared test compares how changes in epsilon change the Hamiltonian calculation. Both the HMC LF and MHMC LF have  $\mathcal{O}(\epsilon^2)$  error scaling. Therefore, if the leapfrog integrators are working correctly, larger values of  $\epsilon^2$  should result in larger changes in the Hamiltonian in a linear fashion. The results of these trials for both the one dimensional and two-dimensional

cases for both HMC and MHMC can be seen below.

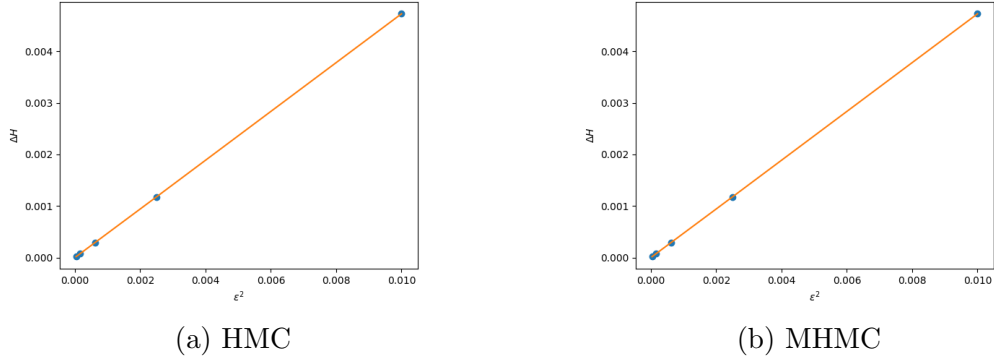


Figure 3.1: QM  $\epsilon^2$  Test

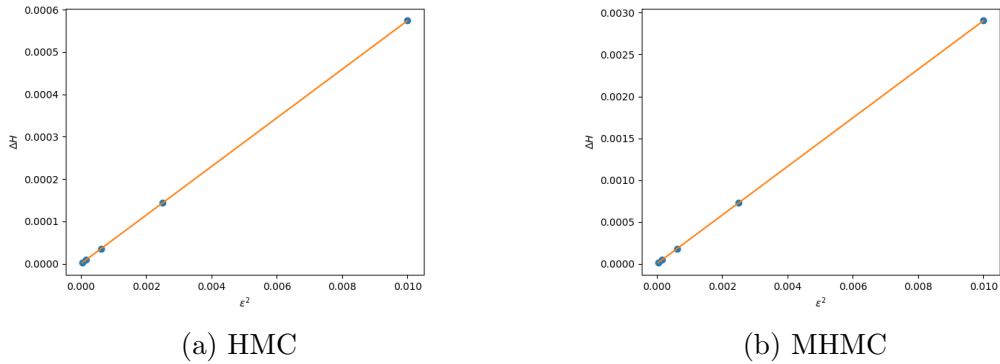


Figure 3.2: SG  $\epsilon^2$  Test

As we can see in 3.1 and 3.2, all combinations of SG, QM, HMC, and MHMC follow a linear  $\epsilon^2$  relationship, thus exhibiting the correctness of the leapfrog integrators.

## 3.2 Reversibility Tests

One of the key tenets of Hamiltonian Dynamics is that the energy is conserved. For both the Quantum Harmonic Oscillator and Sine Gordon models, that means that a change in the Hamiltonian is reversible. A reversibility test works by calculating

a new Hamiltonian after generating a random momentum, then recalculating the Hamiltonian with the sign of the momentum flipped, essentially retracing the path of the first calculation. If reversing the momentum results in the initial Hamiltonian, then energy is conserved in these calculations.

	HMC	MHMC
$H_0$	1061.678357793096	1107.9416409103592
$H_{new}$	1061.6953804315328	1107.9591699410007
$H_{reversed}$	1061.678357793096	1107.941640910141
$H_{reversed} - H_0$	0.0	2.1827872842550278e-10

Table 3.1: QM Reversibility Tests

	HMC	MHMC
$H_0$	5.549393878734056	6.967247948147463
$H_{new}$	5.549404649235499	6.967278021495622
$H_{reversed}$	5.549393878734056	6.96724794814713
$H_{reversed} - H_0$	0.0	-3.3306690738754696e-13

Table 3.2: SG Reversibility Tests

As we can see in 3.1 and 3.2, all combinations of QM, SG, HMC, and MHMC conserve energy. One interesting thing of note, perhaps due to the float storage limitations of python, but MHMC reversibility maintains a very small error term on the magnetic e-10.

### 3.3 QM Tests

In order to test QM using HMC and MHMC we first need to describe. For these tests we use the same parameter for both HMC and MHMC. Due to computational limitations the lattice size is 128. Additionally, the only configuration where either MHMC or HMC get stuck is when  $\omega$  is negative. If omega is positive, both MHMC and HMC sample all possible results equally. The quartic scalar,  $g$ , was set to 1.

The step size,  $\epsilon$  was set to 0.01 with the over trajectory set to 1 as well. This would result in the leapfrog integrator taking 100 steps in its calculations. In order to full run a Monte Carlo simulation, we first have to essentially warm up the algorithm but running it a set number of times in order to work out any outlier that might pop up in the initial few iterations. For these tests, the number of warm up loops was 100. Additionally, we ran each algorithm 1000 while recording the X output of the HMC and MHMC respectively. From these results we can calculated autocorrelation time, compare the magnitude of the outputs and finally directly compare the outputs plotted on the lattice itself. Finally we need a maximum lag in order to calculate the autocorrealtion. For these test we used a maximum lag value of 100.

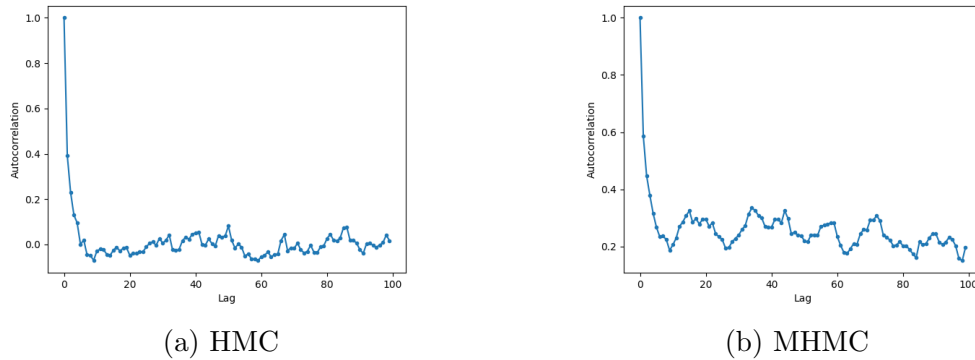


Figure 3.3: QM Autocorrelation

The ideal autocorrelation result is exponential decay. Both HMC and MHMC initially begin as exponential decay. However, the closer the MHMC Autocorrelation gets to 0, the less steep the curve becomes. In fact, MHMC autocorrelation never fully gets down to zero under these parameters. Thus, when directly comparing HMC and MHMC, we can see that HMC actually performs better in the case of the QM model.

Another test to compare the two algorithms is a looking at the sum of  $\sum X_i^2$  for each iteration of the respective algorithm. While not a measure of actual results, by looking at  $\sum X_i^2$  we can compare the magnitude of  $x$  for HMC and MHMC. As we can see in 3.4 both HMC and MHMC and calculate essential the same value for  $\sum X_i^2$ . This result further supports the correctness of MHMC in application and also tells us that we are comparing apples to apples in these experiments.

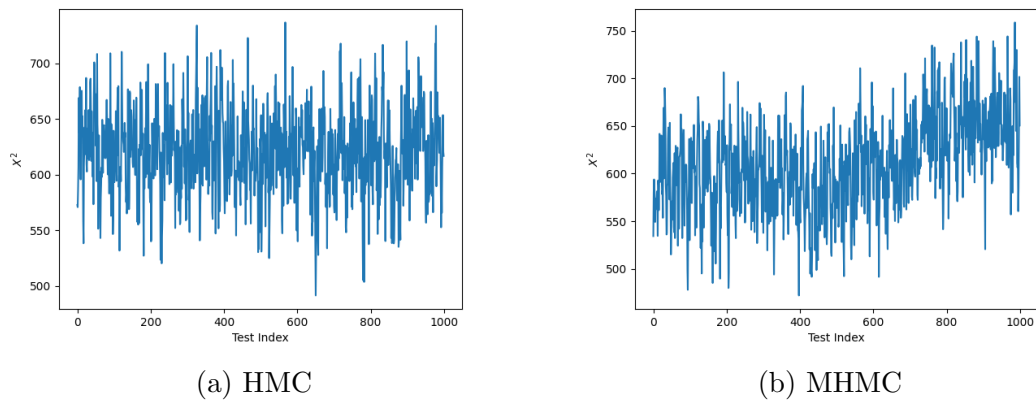


Figure 3.4: QM  $X^2$  Results

The final test in comparing HMC and MHMC in the one-dimensional case is to graph the value of  $x$  on its respective lattice point for every iteration of our test. That is to say plot  $(L,x)$  for all  $L$ . If we then plot these results for each test, we can then see if either algorithm gets stuck in the QM model.

The result of these tests, 3.5 clearly shows that both HMC and MHMC get stuck in every pass through the algorithm. Ideally, we would see continuous sampling across both minimums. However, that is not the case. If anything, HMC performs better than MHMC in this test as well.

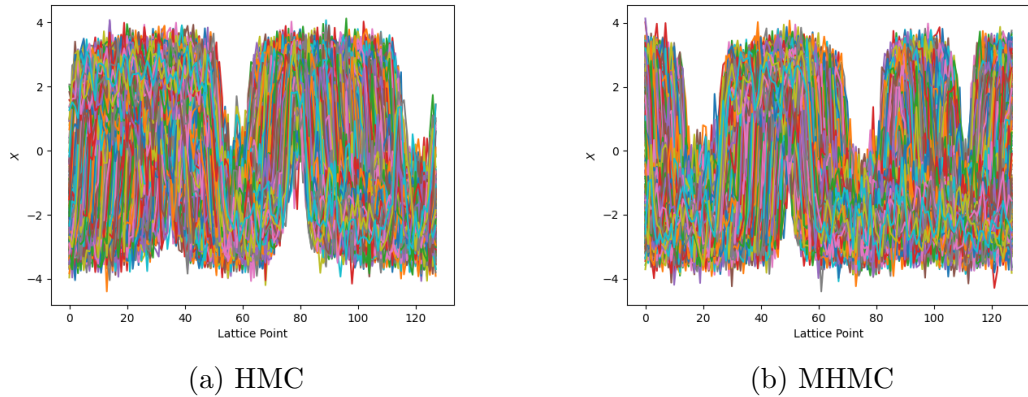


Figure 3.5: QM Results

### 3.4 sine-Gordon Tests

For the sine-Gordon model, the observable we are interested in is the variance. Due to its two dimensional nature, any real attempt to plot  $x$  will result in something unclear. However, the variance allows us to compare the results of HMC with MHMC. However, the most important results is the autocorrelation time. If MHMC is more efficient than HMC, there should be a clear difference in their autocorrelation time.

In testing MHMC on the SG model, we tried three different  $G$  matrices to in an attempt see if there is any discernible difference when it comes to the layout of  $G$  when using MHMC. Due to computational limitations of these calculations, we tested all  $G_1$ ,  $G_2$ , and  $G_3$  on a small  $4 \times 4$  lattice. However, we only tested  $G_3$  on a  $16 \times 16$  and  $32 \times 32$  to see if scaling showed marked differences between HMC and MHMC.

For the sine-Gordon model we one again used the same parameters when testing HMC and MHMC. For our thermalization runs, we looped each algorithm 100 times before saving 1000 runs for analysis. Additionally, all tests used  $\epsilon$  of 0.01. The SG model also requires a temperature component,  $T$ . For these tests, we settled on  $T =$



2. Finally we need a maximum lag in order to calculate the autocorrelation. Like with QM, we used a maximum lag value of 100.

### 3.4.1 4x4 Lattice

#### HMC

As opposed to MHMC, HMC is easily scalable both in lattice size and dimension. For this first test using SG, we are using a small 4x4 lattice.

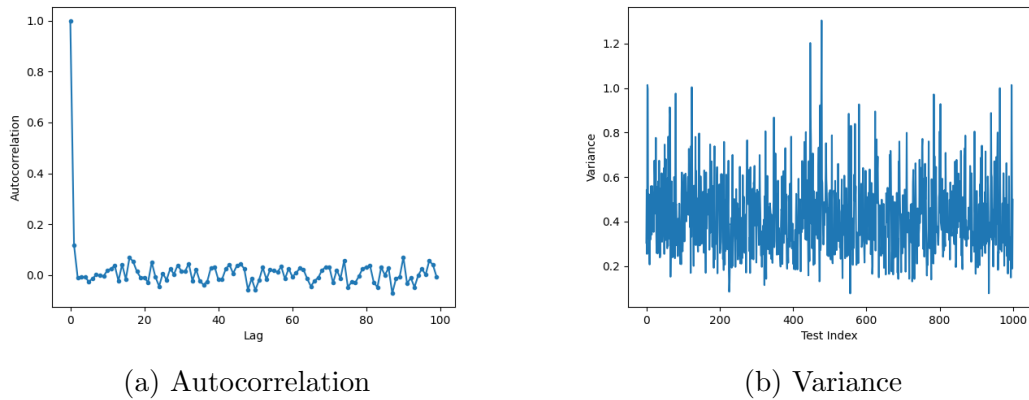


Figure 3.6: SG HMC Small Lattice

As we can see in 3.6a, HMC’s autocorrelation converges on 0 extremely quickly. This shows that it is working efficiently at this small scale despite two-dimensional lattice. Additionally, in 3.6b, we can see that these results are valid due to the mean variance of  $x$  being centered around 0.5.

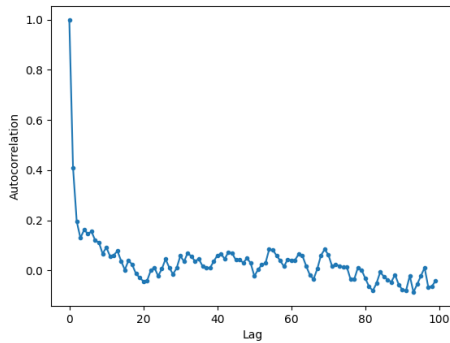
#### MHMC $G_1$

As we’ve discussed previously, the  $G$  matrix is an  $L^2 \times L^2$ . We can then represent this matrix in simpler form as an  $L \times L$  matrix where each matrix element is another  $L \times L$  matrix. This then allows us to approach generating this matrix as a block matrix. The  $G_1$  matrix is thus

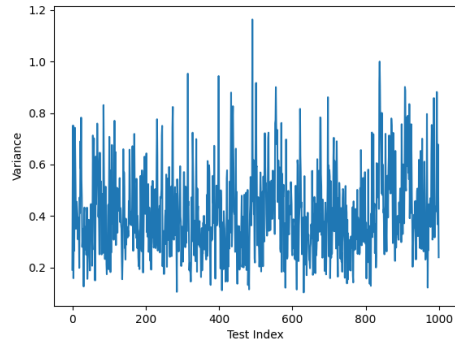
$$G_1 = \begin{bmatrix} B & B & 0 & B \\ B & B & B & 0 \\ 0 & B & B & B \\ B & 0 & B & B \end{bmatrix}$$

where  $B$  is an antisymmetric invertible matrix comprised of off two off diagonal identity matrices of opposite signs or:

$$B = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



(a) Autocorrelation



(b) Variance

Figure 3.7: SG MHMC  $G_1$  4x4 Lattice

As we can see in 3.7a, MHMC does not show any improvement on HMC in terms of autocorrelation time and the speed at which it converges to zero. In fact, when compared to 3.6a, MHMC performs significantly worse.

## MHMC $G_2$

The second  $G$  matrix tried,  $G_2$ , is a slightly less complicated version of  $G_1$  where only only the middle diagonal elements are filled, leaving rest of the matrix empty.

$$G_2 = \begin{bmatrix} B & B & 0 & 0 \\ B & B & B & 0 \\ 0 & B & B & B \\ 0 & 0 & B & B \end{bmatrix}$$

For  $G_2$  the interior matrix elements,  $B$ , are the same as in  $G_1$  thus maintaining  $G_2$ 's invertible and antisymmetric properties.

The  $G_2$  using a small lattice results are similar to that of the  $G_1$  matrix. By extension both tests fail to improve upon the HMC alorithm. When comparing autocorrelation time, HMC, 3.6a shows a much quicker convergence in autocorrenalntion time when compared to 3.8a. In comparing the variances HMC and MHMC, 3.6b and 3.8b respively, both nicely oscillate around a mean of 0.5, thus showing that both HMC and MHMC using the  $G_2$  matrix are calculating similar distributions. However, HMC is simply more efficient than MHMC using  $G_2$ .

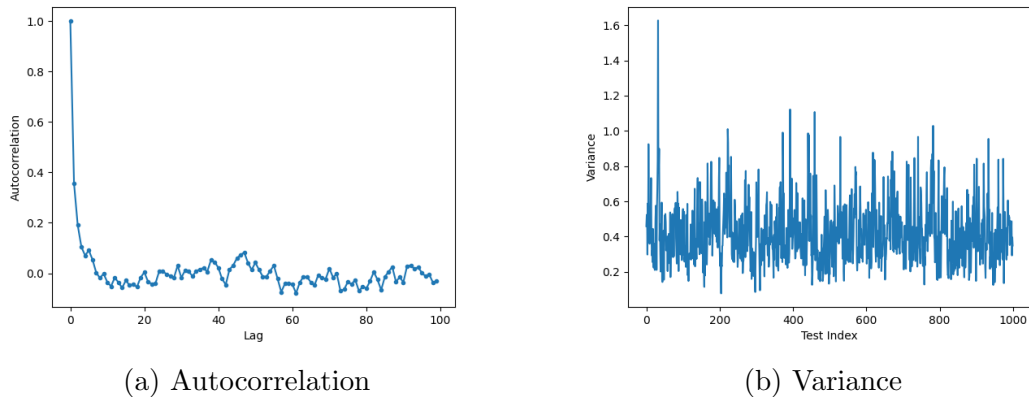


Figure 3.8: SG MHMC  $G_2$  4x4 Lattice

As we can see in 3.8a, MHMC does not show any improvement on HMC in terms

of autocorrelation time and the speed at which it converges to zero. However, its autocorrelation distribution does show a minor improvement when compared to the 3.7a. This suggests that changes in  $G$  do have positive effects when it comes to MCMC convergence.

### MHMC $G_3$

For the  $G_3$  matrix we took a slightly different approach in choosing the matrix. Rather than, using variation of the identity matrix only inside each matrix element. We constructed a matrix where the larger  $L^2, L^2$  structure is the same as the  $B$  matrix outlined above. The middle diagonal was then filled with  $B$  matrices as well.  $G_3$ , as a 4x4 matrix, is

$$G_3 = \begin{bmatrix} B & -A & 0 & 0 \\ A & B & -A & 0 \\ 0 & A & B & -A \\ 0 & 0 & A & B \end{bmatrix}$$

where

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

3.9a shows that  $G_3$  performs the worst out of all three  $G$  matrices. Once again confirming our suspicion that an there is an ideal  $G$  matrix. Like all the calculations so far, the variance for each iteration oscillates around a mean of approximately 0.5 which is expected.

For the larger lattice tests, detailed below, we choose to scale  $G_3$  as use it in the MHMC calculations. Due to its poor performance in the 4x4 lattice, it has the most

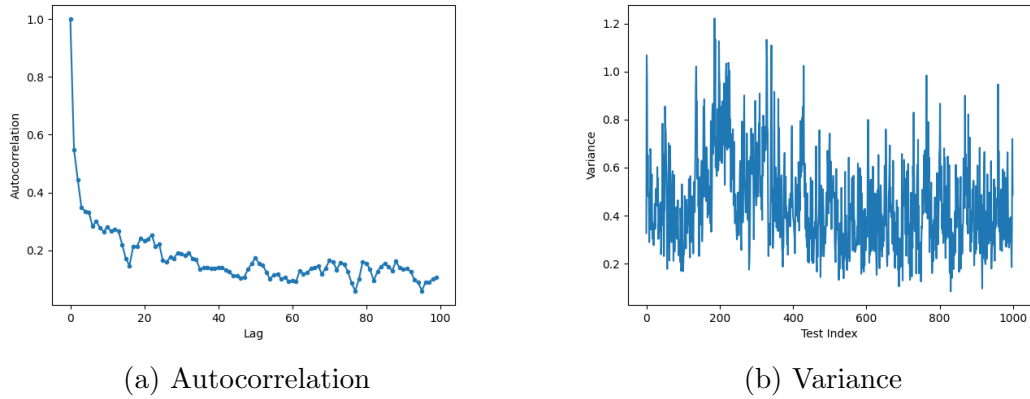


Figure 3.9: SG MHMC  $G_3$  4x4 Lattice

room for improvement and thus might point to whether lattice size changes MHMC efficiency in comparison to HMC.

### 3.4.2 16x16 Lattice

#### HMC

HMC is very easily scaled to larger lattice sizes. For this test we called up the lattice size to where  $L = 16$ . With this larger lattice, 3.10a shows a slower convergence to 0 than in the smaller lattice, 3.6a. On the other hand the variance for iteration through the lattice stayed essentially the same as shown by 3.10b and 3.6b.

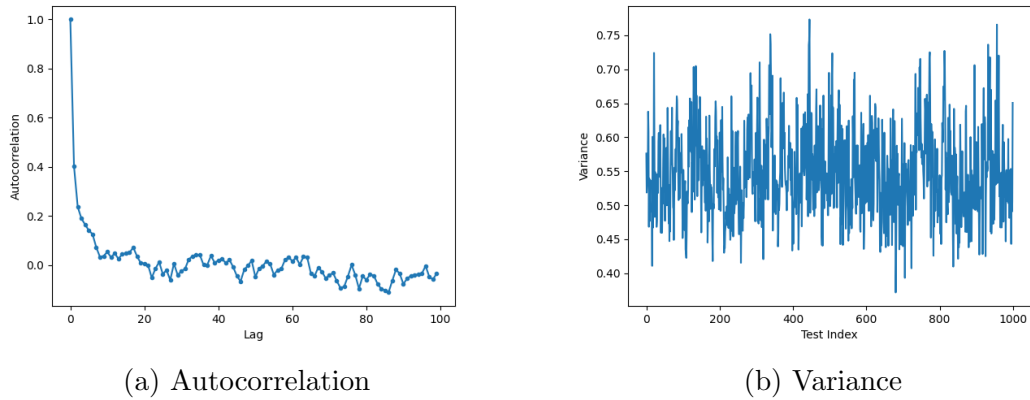


Figure 3.10: SG HMC 16x16 Lattice Tests

## MHMC

MHMC, in comparison, shows a similarly slow convergence in autocorrelation time as seen in 3.11a. At least for the 16x16 Lattice using  $G_3$ , MHMC does not show any advantage over HMC in terms of convergence time. On the other hand, both algorithms, despite their less than stellar autocorrelation times, compute the same variance, showing once again that the collocations are correct, but simply not efficient.

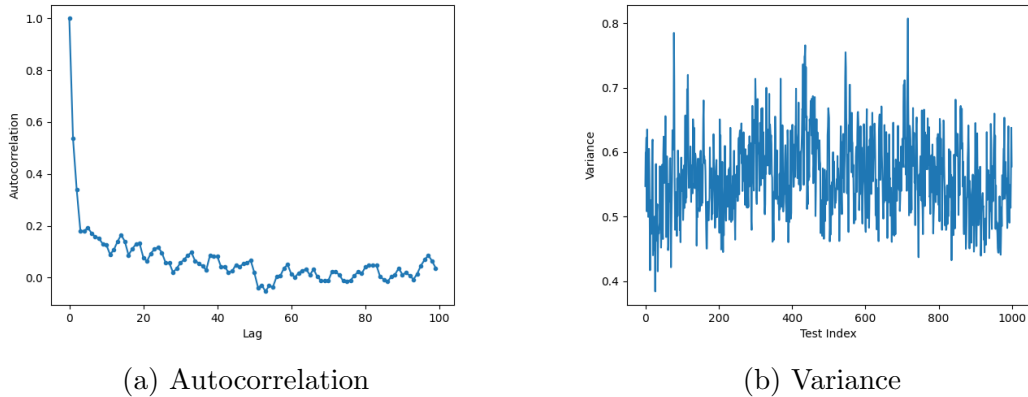


Figure 3.11: SG MHMC  $G_3$  16x16 Lattice Tests

### 3.4.3 32x32 Lattice

The final test that we compute was comparing MHMC using  $G_3$  to HMC on a much larger 32x32 sized lattice. This calculation takes multiple minutes with computational power we had available at the time and served as a sort of upper boundary in Lattice size. Any attempts using Lattices larger than this resulted in computations taking many hours with no results.

## HMC

For HMC, seen in 3.12, we see a rather consistent autocorrelation plot that shows it swiftly approaching zero as the lag increases. Additionally the variance once against oscillates around the mean as expect.

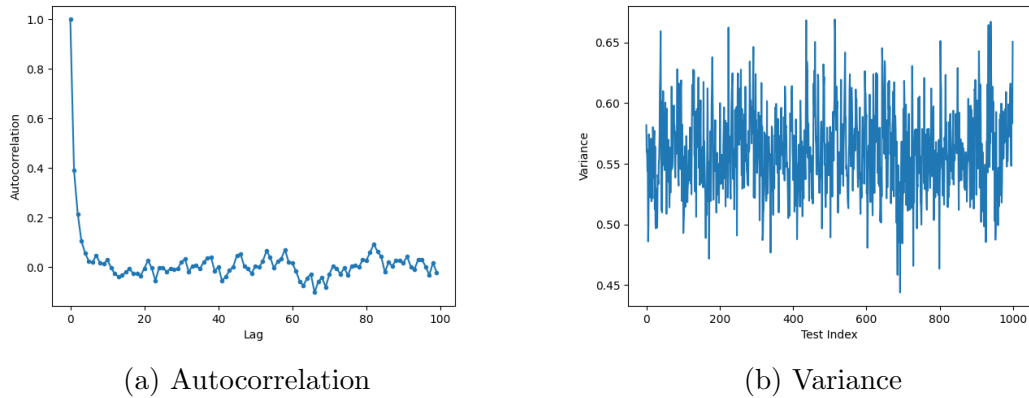
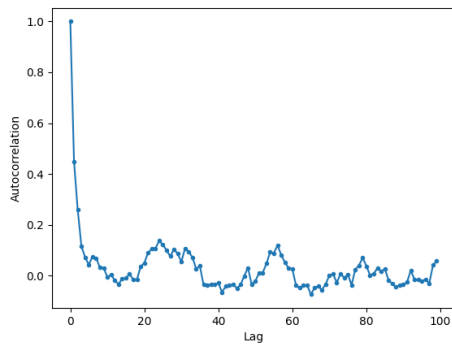


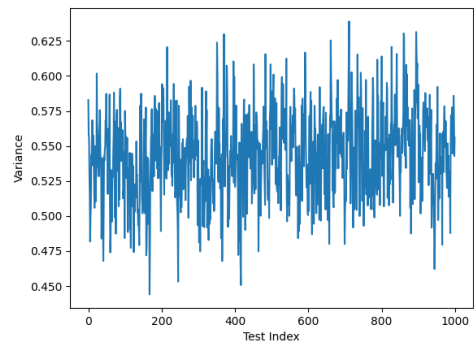
Figure 3.12: SG HMC 32x32 Lattice Tests

## MHMC

MHMC (3.13), using  $G_3$ , shows what appears to be slight improvement on its previous results for this larger lattice. However, the lag increases we see rather large spikes in autocorrelation time where HMC does not. This suggests that it is still less efficient than HMC at this large lattice size. As ever, the variance oscillates nicely around the mean as we would expect.



(a) Autocorrelation



(b) Variance

Figure 3.13: SG MHMC  $G_3$  32x32 Lattice Tests



# Chapter 4

## Conclusion/Outlook

While an interesting proposal, MHMC fails to provide any increase in efficiency in our exploration of the algorithm. In the one-dimensional case, 3.5, both HMC and MHMC get stuck and fail to sample the entire distribution. Additionally, when looking at the autocorrelation function in particular 3.3, HMC shows itself to be more efficient than MHMC. This result continues when testing MHMC on SG. HMC outperforms MHMC regardless of lattice size or the chosen  $G$  matrix as seen in 3.7a, 3.8a, 3.9a, 3.11a, and 3.13a. One caveat to this result is the broad definition of the  $G$  matrix. While we tested three different  $G$  matrices in the SG case, the only restrictions for these matrices are that they are invertible and anti-symmetric. As the lattice size increases, the number of possible  $G$  matrices increases. For our experiments we were limited in computational power, and thus had to use relatively small matrices for the SG tests. Consequently, MHMC's efficiency could scale differently to HMC when the lattice gets large. Additionally, our computational limitations meant that we only tested MHMC on a one-dimensional or two-dimensional model. Just like how it could scale differently with lattice size, MHMC might prove itself more efficient as the number of dimensions increase. Finally, it is not unimaginable that there is a  $G$  matrix for a large lattice that makes MHMC more efficient than HMC. This would perhaps require some form of neural network to generate a more efficient  $G$  matrix.

# Appendix A

## Example Leapfrog Integrators

These are examples of the integrators used in for this research project. Of note, the force function is model dependent.

### A.1 HMC Integrator

The following is the python code which computes the leapfrog integration for the Hamiltonian Monte Carlo algorithm. This could also be described as the canonical leapfrog integrator. This works for both the one-dimensional and two dimensional case since  $p$  and  $x$  have the same dimensions.

```
/*  
p = p + 0.5*force(x)*eps  
for t in range(1,nsteps):  
    x = x + p*eps  
    p = p + force(x)*eps  
x = x + p*eps  
p = p + 0.5*force(x)*eps  
*/
```

### A.2 MHMC One-Dimensional Integrator

The MHMC leapfrog is more complicated. However, the clear combination between new terms containing the  $G$  matrix and canonical calculations can be seen. Despite  $G$  being an  $L \times L$  matrix, no reshaping needs to happen for the one-dimensional case due to the nature of the `numpy.dot()` function.

```
/*  
G_exp = expm(G*eps)  
G_inv = inv(G)  
p = p = p + 0.5*force(x)*eps  
for n in range(1, self.nsteps):  
*/
```

```

    x = x + np.dot(G_inv,np.dot(G_exp-np.identity(len(G)),p))
    p = np.dot(G_exp,p)
    p = p + 0.5*force(x)*eps
x = x + np.dot(G_inv,np.dot(expm(G*self.eps)-np.identity(len(G)),p))
p = np.dot(G_exp,p)
p = p + 0.5*force(x)*eps

```

### A.3 MHMC Two-Dimensional Integrator

The two-dimensional integrator requires reshaping in order to use the `numpy.dot()` matrix multiplication function. However, after flattening the momentum matrix in order to multiply both the  $G$  exponential and  $G$  inverse, it must be reshaped back into an  $L \times L$  matrix in order to calculate the canonical change in momentum or the MHMC change in  $x$ .

```

/*****/

G_exp = expm(G*self.eps)
G_inv = inv(G)
p = p + 0.5*force(x)*eps
for n in range(1, self.nsteps):
    x = x + np.reshape((np.dot(G_inv,
        np.dot(G_exp-np.identity(len(G)),np.matrix.flatten(p)))),(self.L,self.L))
    p = np.reshape(np.dot(G_exp,np.matrix.flatten(p)),(self.L,self.L))
    p = p + 0.5*force(x)*eps
x = x + np.reshape((np.dot(G_inv,
np.dot(G_exp-np.identity(len(G)),np.matrix.flatten(p)))),(self.L,self.L))
p = np.reshape(np.dot(G_exp,np.matrix.flatten(p)),(self.L,self.L))
p = p + 0.5*force(x)*eps

```

# Appendix B

## Autocorrelation

### B.1 Example Autocorrelation Function

This is an example of the autocorrelation function we used for this experiment.  $T$  is the lag value and length refers to the length of the list. As mentioned in 2.6, we calculate an autocorrelation value for the range of lag values and sum over these values to create one autocorrelation value for each list.

```
/******  
mean = np.mean(list)  
d = list - mean  
cor = np.empty([T])  
error = np.empty([T])  
for n in range(T):  
    sd = np.roll(d, -n)  
    g = d[:length-n]*sd[:length-n]  
    cor[n] = np.mean(g)  
    error[n] = np.std(g)/np.sqrt(g.shape[0]-1)  
cor = cor/cor[0]  
error = error/cor[0]
```

# Bibliography

- [1] Lapage, Peter G. (2005). Lattice QCD for Novices. Proceedings of HUGS 98, edited by J.L. Goity, World Scientific (2000)
- [2] Tripuraneni, N., Rowland, M., Ghahramani, Z., Turner, R. (2017). Magnetic Hamiltonian Monte Carlo, Proceedings of the 34th International Conference on Machine Learning, in PMLR 70:3453-3461
- [3] Sinharay, S. (2003), ASSESSING CONVERGENCE OF THE MARKOV CHAIN MONTE CARLO ALGORITHMS: A REVIEW. ETS Research Report Series, 2003: i-52.
- [4] van Ravenzwaaij, D., Cassey, P., Brown, S.D. (2018), A simple introduction to Markov Chain Monte-Carlo sampling. Psychon Bull Rev 25, 143–154 .