Exploring the Phase Structure of a Unitary Matrix Model in an External Field

(Project Report)

Department of Physical Sciences
Indian Institute of Science Education and Research (IISER)
Mohali

July 17, 2021

Author -Sayar Mandal *Supervisor - Dr. Anosh Joseph*

Abstract

In this project we have tried to numerically simulate and verify the analytical result for the Polyakov loop observable for a unitary matrix model using Monte Carlo method. Our simulations verify that this unitary matrix model, which is a deformation of the Gross-Witten-Wadia (GWW) model, also undergoes a phase transition at the same critical coupling as that of the GWW model.

Acknowledgements

I would like to express my sincerest gratitude to Dr. Anosh Joseph for giving me the opportunity to work on this project and for his constant support and guidance to help me understand the topic and complete the project in time. I would also like to thank Navdeep Singh Dhindsa for his suggestions on implementing the Markov chain Monte Carlo (MCMC) algorithm. And, last but not least, I would like to thank my peers James Watt, Shantanu Raikwar, Tushar Baruah, and senior Roshan Kaundinya for the insightful discussions on the topic.

Sayar Mandal
IISER Mohali
July 17, 2021

Contents

1 Introduction

The first section of this report elaborates on the MCMC method, Metropolis algorithm and QR algorithm, which were integral parts of the numerical simulation of the unitary matrix model. Following this, we have set up the mathematical formalism for the unitary matrix model and its deformed counterpart. Finally, we have verified the analytical result for the Polyakov loop observable presented in [1] and [2].

2 Markov Chain Monte Carlo (MCMC)

MCMC comes to the rescue when we might need to sample a probability distribution (possibly in a very high dimension), which is implausible to do analytically. The basic idea behind MCMC is to perform a "random walk" through the probability distribution under study favoring the points with higher probability.

2.1 Stationary Distribution

In a Markov chain the state of the system solely depends on the previous state. We have

$$s_{n+1} = s_n P, \tag{1}$$

where P is the transition matrix and s_n is the distribution in the n^{th} step. A distribution π is defined as stationary distribution if it satisfies

$$\pi = \pi P. \tag{2}$$

2.2 Ergodicity

We define $P_{i,j}^{(n)}$ to be the probability of reaching state j from state i in n steps. A Markov chain with M possible states is called ergodic if for some n

$$P_{i,j}^{(n)} > 0 \quad \forall\ i, j = 0, 1, \cdots, M. \tag{3}$$

Whenever the condition of ergodicity is satisfied, the Markov Chain always has a unique stationary distribution and it is the same as the limiting distribution *i.e.*

$$\pi = s_n = \lim_{n\to\infty} s_0 P^{(n)}. \tag{4}$$

Ergodicity is the combination of the "irreducible" and "aperiodic" properties of a Markov chain. Irreducible implies that for every state there is a positive probability

to move to another state. Aperiodic implies there is a positive probability to move to any state in the system from any other state, or in other words the chain cannot get trapped in cycles. During the "random walk" in Markov chain we must be sure that the system can reach all states with frequency proportional to their respective probabilities. Hence for MCMC to work the Markov chain must be ergodic.

2.3 Metropolis Algorithm

In Ref. [3] the authors summarized what would be known as Metropolis algorithm, and later it was generalized to the Metropolis-Hastings Algorithm. The algorithm is as follows

- Let one molecule move slightly.
- Calculate the change in energy ΔE of the system.
- If it is less than the energy of the previous state, accept it.
- Otherwise accept the change with a probability of $e^{-\frac{\Delta E}{k_B T}}$, where k_B is the Boltzmann constant and T is the temperature of the system.

2.4 Practical Nuances

2.4.1 Thermalization

We have used the Metropolis algorithm to sample from a Boltzmann distribution. This algorithm is iterative, and so we cannot directly use the action values generated in the very first few thousand steps as it will not give us the Boltzmann distribution. This is because of the reason that the action is still evolving to the equilibrium distribution. These first few thousand steps before action is equilibrated are called "thermalization" steps. We need to do some trial and error to find the required number of thermalization steps.

2.4.2 Acceptance Rate

Acceptance rate is calculated as the number of accepts per 100 sweeps. A very high or very low acceptance rate indicates that the step size is not optimized. A good rule of thumb is $(40 - 90)\%$ acceptance.

3 QR Algorithm

One of the important steps in simulating the unitary matrix model is re-unitarization after each sweep. To do so we need to employ the *QR decomposition*. We also need to calculate the eigenvalues to find the parameter s, which is done using the QR algorithm.

3.1 Householder Reflections

Although there are different ways to perform the QR decomposition, Householder reflections is the one popularly used by different libraries for various programming languages.

We discuss this below.

Let $\vec{x}$ be a vector reflected in a mirror (plane/hyperplane) with a normal vector $\vec{v}$. The orthogonal projection of $\vec{x}$ on $\vec{v}$ is given by

$$\frac{x^T v}{\|v\|}\frac{v}{\|v\|} = \frac{x^T v}{v^T v}v. \tag{5}$$

Since the mirror is perpendicular to $\vec{v}$, $x - (\frac{x^T v}{v^T v})v$ lies on the plane of the mirror.

Since reflections are equidistant from the mirror, the reflected point can be written as

$$x - 2\frac{x^T v}{v^T v}v = x - 2\frac{v^T x}{v^T v}v = (I - 2\frac{vv^T}{v^T v})x, \tag{6}$$

where $Q_v = (I - 2\frac{vv^T}{v^T v})$ is the *Householder reflection* matrix.

3.2 QR Decomposition

For the QR decomposition of an $N \times N$ matrix A, we find an orthogonal matrix Q and an upper diagonal matrix R such that $A = QR$. Householder's idea was to multiply the Householder reflections to the left of A to gradually convert A to an upper triangular matrix.

If the first column of the matrix A is $\vec{a}$, we want to find a vector $\vec{v}$ such that $Q_v a = \|a\|e_1$, where $e_1 = [1, 0, 0, \cdots]^T$. By doing so we can remove $N - 1$ elements from the first column of A. This is the first step to construct the upper triangular form of A. It turns out that $v = a - sign(a)\|a\|e_1$, where $sign(a)$, which is the sign of the first entry of $\vec{a}$, solves the condition stated above. After the first step we have

$Q_v = Q_1$ such that

$$Q_1 A = \left[\begin{array}{c|ccc} a_{11}^{'} & a_{12}^{'} & \cdots & a_{1N}^{'} \\ \hline 0 & & & \\ \vdots & & A' & \\ 0 & & & \end{array}\right]. \tag{7}$$

For the next step we construct

$$Q_2 = \left[\begin{array}{c|ccc} 1 & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & Q_2^{'} & \\ 0 & & & \end{array}\right], \tag{8}$$

where $Q_2^{'}$ is a $N-1$ dimensional matrix constructed using the same idea as Q_1, with respect to $A^{'}$ to zero out the last $N-2$ elements of the first column of $A^{'}$. We have

$$Q_2 Q_1 A = \left[\begin{array}{cc|ccc} a_{11}^{''} & a_{12}^{''} & a_{13}^{''} & \cdots & a_{1N}^{'} \\ 0 & a_{22}^{''} & a_{23}^{''} & \cdots & a_{2N}^{''} \\ \hline 0 & 0 & & & \\ \vdots & \vdots & & A'' & \\ 0 & 0 & & & \end{array}\right]. \tag{9}$$

For the k^{th} transformation matrix

$$Q_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & Q_k^{'} \end{bmatrix}, \tag{10}$$

where $Q_k^{'}$ is a $(N-k+1)$ dimensional matrix based on the first column of the sub-matrix A_{k-1} of same dimensions obtained in the previous step.

After $(N-1)$ iterations A becomes upper triangular

$$R = Q_{N-1} \cdots Q_3 Q_2 Q_1 A. \tag{11}$$

We obtain Q as $(Q_{N-1} \cdots Q_3 Q_2 Q_1)^{-1}$.

3.3 QR Algorithm

Two square matrices A and B are similar if

$$A = C^{-1} B C, \tag{12}$$

where C is an invertible matrix.

Let us say we want to find the eigenvalues of a matrix A_0. In the k^{th} step, using the above described QR decomposition method, we get

$$A_k = Q_k R_k. \tag{13}$$

Then we define

$$A_{k+1} = R_k Q_k = Q_k^{-1} A Q_k. \tag{14}$$

Therefore, all the A_k are similar and have the same eigenvalues.

Starting from 0 if we iterate over the steps described in Eqs. (13) and (14), the A_k matrix will converge to a diagonal form with eigenvalues for A_k as the elements in the diagonal.

4 Gross-Witten-Wadia (GWW) Model

The Gross-Witten-Wadia (GWW) model is a unitary matrix model. The action is given by

$$S_{GWW} = -\frac{1}{g^2}\mathrm{Tr}(U + U^\dagger), \tag{15}$$

where g is the coupling factor and U is an $N \times N$ unitary matrix.

We have the Hooft's coupling $\lambda = g^2 N$. Thus the action becomes

$$S_{GWW} = -\frac{N}{\lambda}\mathrm{Tr}(U + U^\dagger). \tag{16}$$

The partition function is given by

$$Z = \int DU \; e^{-S_{GWW}}. \tag{17}$$

Instead of using U directly if we work with the angle variables θ_i , $\forall i = 1, 2, 3, \cdots, N$, then the matrix U takes the diagonal form

$$U = D = \begin{bmatrix} e^{i\theta_1} & 0 & 0 & \cdots & 0 \\ 0 & e^{i\theta_2} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & e^{i\theta_N} \end{bmatrix}, \qquad -\pi < \theta_i \leq \pi. \tag{18}$$

The change of variables gives rise to a Jacobian, known as the *Vandermonde determinant*. Thus we have the action

$$
\begin{aligned}
S_{GWW} &= -\sum_{j,k=1,j\neq k}^{N} \ln\sin\left|\frac{\theta_j-\theta_k}{2}\right| - \frac{N}{\lambda}\mathrm{Tr}(D+D^\dagger) \\
&= -\sum_{j,k=1,j\neq k}^{N} \ln\sin\left|\frac{\theta_j-\theta_k}{2}\right| - \frac{N}{\lambda}\sum_{k=1}^{N}(e^{i\theta_k}+e^{-i\theta_k}).
\end{aligned} \tag{19}
$$

4.1 Polyakov loop

The analytical solution to the Polyakov loop observable is given by

$$
P = \begin{cases} 1-\frac{\lambda}{4} & \text{for } \lambda < 2 \\ \frac{1}{\lambda} & \text{for } \lambda \geq 2. \end{cases} \tag{20}
$$

The Polyakov loop observable was calculated from MCMC data using

$$
P = \frac{1}{N}\sum_{k=1}^{N} e^{(i\theta_k)}. \tag{21}
$$

4.2 Code Snippet

The following pseudo-code explains the implementation of Metropolis algorithm for the GWW model using Eq. (19). To simulate the GWW model the "random walk" was performed on the space θ_i , $\forall i = 1, 2, 3, \cdots, N$. We used $N = 10$ and step size = 0.5. For each data point 1000 thermalization steps and 10000 sweeps were performed.

```
import numpy as np
import random

#Initialization
theta_old = 0.01*np.random.rand(N) + 0.01*j*np.random.rand(N)
theta_new = np.copy(theta_old)

#Metropolis Update
old_action = get_action(theta_old, g)
change_index = np.random.randint(0, N)
noise = 2 * (random.random() - 0.5) + 0.0j
theta_new[change_index] = theta_old[change_index] + step_size* noise
```

```
new_action = get_action(theta_new, g)
delta_S = new_action - old_action
u = random.random()

#Accept-Reject
if (np.exp(-delta_S) >= u or delta_S < 0):
    accept += 1
    theta_old[change_index] = theta_new[change_index]

else:
    theta_new[change_index] = theta_old[change_index]
    new_action = np.copy(old_action)
```

4.3 Simulation Results

From Fig. 1 we see that the MCMC data and analytical solution for the Polyakov loop observable are in in good agreement.

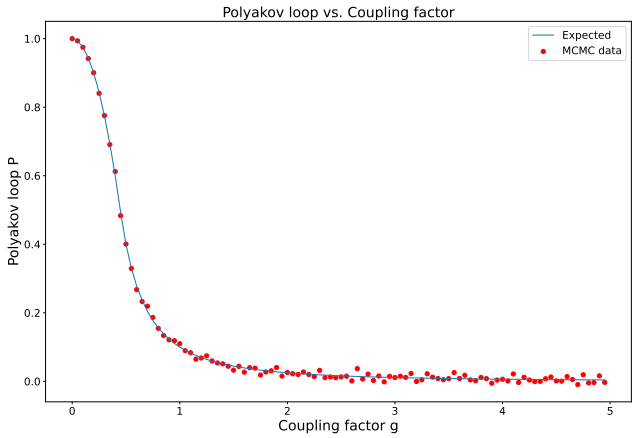


Figure 1: Comparison of the numerical results with the analytical result for the Polyakov loop observable.

5 Unitary Matrix Model in an External Field

In this section we will consider a deformation of the GWW model. We have the action

$$S = -\mathrm{Tr}\left(UA^\dagger + AU^\dagger\right), \tag{22}$$

where U is an $N \times N$ unitary matrix and A is a $N \times N$ arbitrary complex matrix with real and positive eigenvalues.

We define a matrix

$$H = AA^{\dagger}. \tag{23}$$

If we take λ_a as the eigenvalues of the H matrix we can define a parameter s as

$$s \equiv \mathrm{Tr}(H^{-1/2}) = \sum_{a=1}^{N} \left(\frac{1}{\sqrt{\lambda_a}} \right). \tag{24}$$

When $s > 2$ the model is in the strong coupling regime and when $s < 2$ the model is in the weak coupling regime.

5.1 GWW Model as the Special Case

If we consider the case

$$H = \begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & \lambda_N \end{bmatrix} \tag{25}$$

with λ_a real and positive, then

$$A = A^{\dagger} = \begin{bmatrix} \sqrt{\lambda_1} & 0 & 0 & \cdots & 0 \\ 0 & \sqrt{\lambda_2} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & \sqrt{\lambda_N}. \end{bmatrix} \tag{26}$$

The action takes the form

$$\begin{aligned} S &= -\mathrm{Tr}(UA^{\dagger} + AU^{\dagger}) \\ &= -\sum_{j,k=1, j\neq k}^{N} \ln \sin \left| \frac{\theta_j - \theta_k}{2} \right| - \mathrm{Tr}(DA^{\dagger} + AD^{\dagger}) \\ &= -\sum_{j,k=1, j\neq k}^{N} \ln \sin \left| \frac{\theta_j - \theta_k}{2} \right| - \sum_{a=1}^{N} \sqrt{\lambda_a}(e^{i\theta_a} + e^{-i\theta_a}). \end{aligned} \tag{27}$$

Let us take $\lambda_a = \frac{N^2}{\lambda^2}$. Then

$$s = \sum_{a=1}^{N} \frac{1}{\sqrt{\lambda_a}} = \sum_{a=1}^{N} \frac{\lambda}{N} = \lambda. \tag{28}$$

The action becomes

$$S = -\sum_{j,k=1, j\neq k}^{N} \ln \sin \left| \frac{\theta_j - \theta_k}{2} \right| - \frac{N}{\lambda} \sum_{a=1}^{N} (e^{i\theta_a} + e^{-i\theta_a}). \tag{29}$$

Thus the GWW model is a special case of the unitary matrix model in external field in which

$$A = \frac{N}{\lambda} \mathbf{I}_{N\times N}. \tag{30}$$

The phase transition occurs at

$$s = \lambda = g^2 N = 2. \tag{31}$$

5.2 Code Snippet

5.2.1 GWW Model

The following pseudo-code explains the implementation of Metropolis algorithm for the GWW Model. For the simulations we used $N = 20$ and step size was increased from 0.093 to 1.893 as λ varied from 0.01 to 4.76. For each data point 20000 thermalization steps and 100000 sweeps were used. As discussed before

$$A = \frac{N}{\lambda} \mathbf{I}_{N\times N} \tag{32}$$

was used. The "random walk" was performed on the elements of U matrix.

```
import numpy as np
from numpy.linalg import qr
import random
from mpmath import mp

#Initialization
A = np.eye(N) * (N / Lambda)
U = np.eye(N) + 0.01*(np.random.rand(N, N) + np.random.rand(N, N)*1j)
U, _ = qr(U, mode="complete")
```

```
#Metropolis Update
old_action = gen_action(U, A)
U_new = np.copy(U)

index_a = np.random.randint(0, N)
index_b = np.random.randint(0, N)
noise = 2*(random.random() - 0.5) + 2*(random.random() - 0.5)*1j
U_new[index_a][index_b] += step_size * noise
U_new, _ = qr(U_new, mode="complete")
new_action = gen_action(U_new, A)

delta_S = new_action - old_action
u = random.random()

#Accept-Reject
if mp.exp(-delta_S) >= u or delta_S < 0:
    accept += 1
    U = np.copy(U_new)
    if is_therm == 0:
        action.append(new_action)
    else:
        action_therm.append(new_action)
else:
    if is_therm == 0:
        action.append(old_action)
    else:
        action_therm.append(old_action)
```

5.2.2 Unitary Matrix Model in an External Field

The following pseudo-code explains the implementation of Metropolis algorithm for unitary matrix model in an external field. We used $N = 20$ and step size $= f(\lambda)$. For each data point 20000 thermalization steps and 100000 sweeps were used. As discussed before A was set as a random diagonal matrix. The "random walk" was performed on the elements of U matrix.

```
import numpy as np
from numpy.linalg import qr
import random
from mpmath import mp

#Initialization
```

```
A = (np.eye(N, N) + np.diag(np.diag(np.random.rand(N, N))))*(N/Lambda)
U = np.eye(N) + 0.01*(np.random.rand(N, N) + np.random.rand(N, N)*1j)
U, _ = qr(U, mode="complete")

#Metropolis Update
old_action = gen_action(U, A)
U_new = np.copy(U)

index_a = np.random.randint(0, N)
index_b = np.random.randint(0, N)
noise = 2*(random.random() - 0.5) + 2*(random.random() - 0.5)*1j
U_new[index_a][index_b] += step_size * noise
U_new, _ = qr(U_new, mode="complete")
new_action = gen_action(U_new, A)

delta_S = new_action - old_action
u = random.random()

#Accept-Reject
if mp.exp(-delta_S) >= u or delta_S < 0:
    accept += 1
    U = np.copy(U_new)
    if is_therm == 0:
        action.append(new_action)
    else:
        action_therm.append(new_action)
else:
    if is_therm == 0:
        action.append(old_action)
    else:
        action_therm.append(old_action)
```

5.3 Polyakov Loop

Since this is a matrix based approach we calculate Polyakov loop as

$$P = \frac{1}{N}\mathrm{Tr}(U). \tag{33}$$

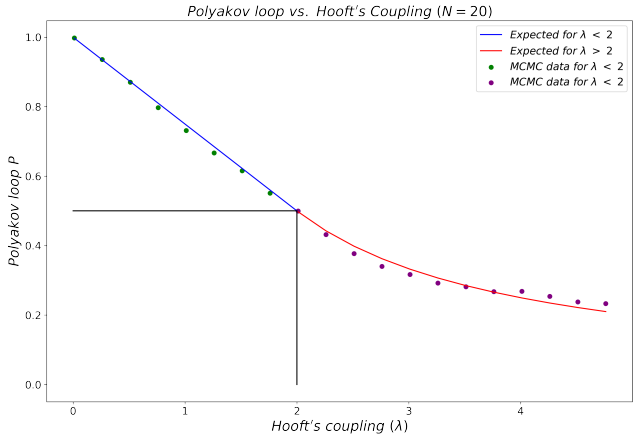
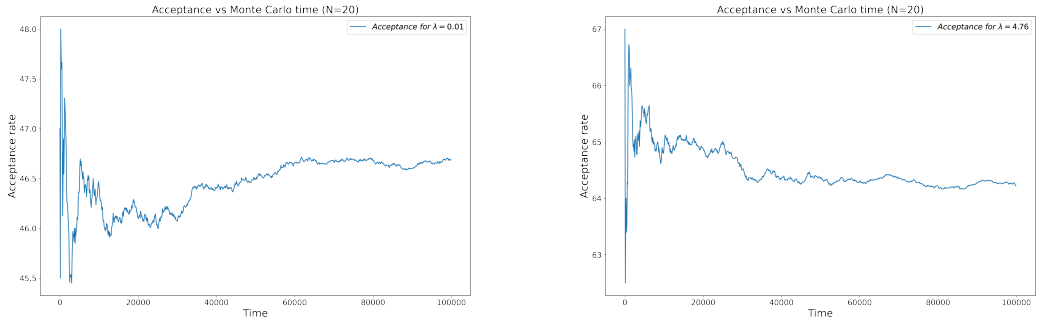


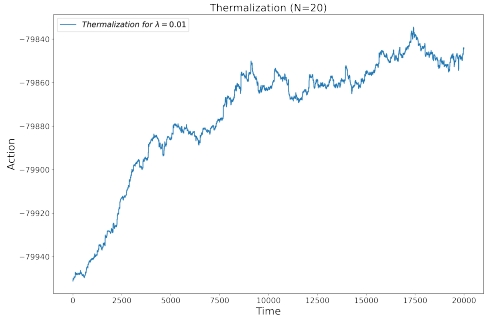
Figure 2: Comparison of simulated values of Polyakov loop with analytical value for the GWW model.

5.3.1 GWW Model

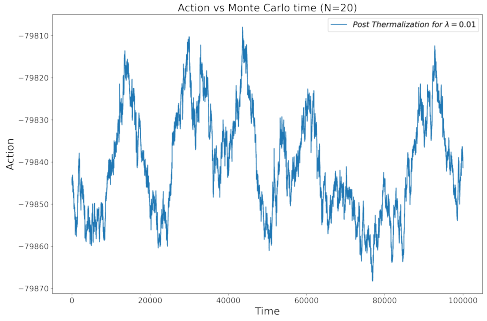


(a) Acceptance rate for $\lambda = 0.01$ (b) Acceptance rate for $\lambda = 4.76$

Figure 3: Acceptance rate for the MCMC simulation of the GWW model.

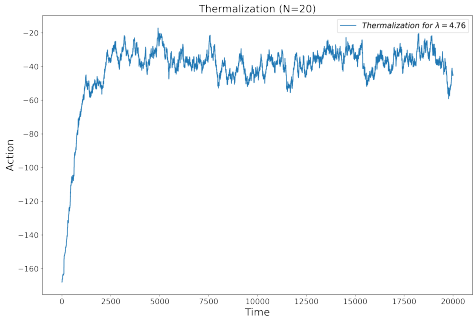


(a) Thermalization for $\lambda = 0.01$

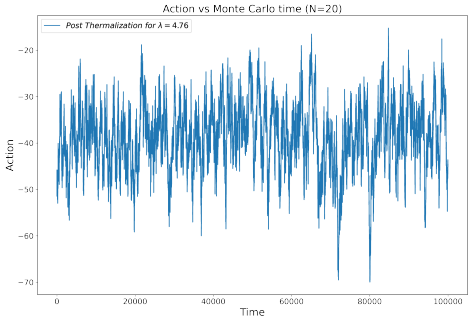


(b) Action for $\lambda = 0.01$

Figure 4: Thermalization and action for the MCMC simulation of the GWW model for $\lambda = 0.01$.



(a) Thermalization for $\lambda = 4.76$



(b) Action for $\lambda = 4.76$

Figure 5: Thermalization and action for the MCMC simulation of the GWW model for $\lambda = 4.76$.

5.3.2 Unitary Matrix Model in an External Field

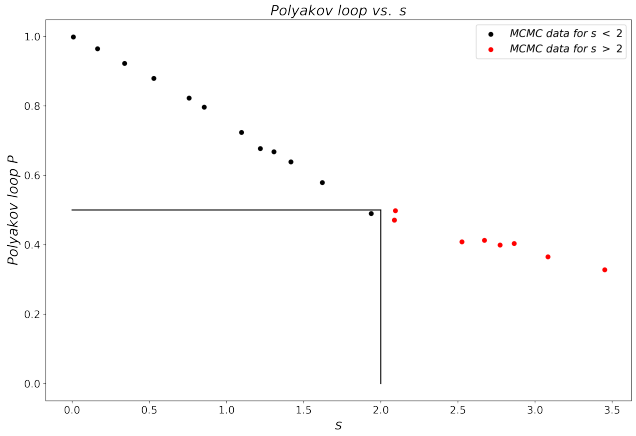
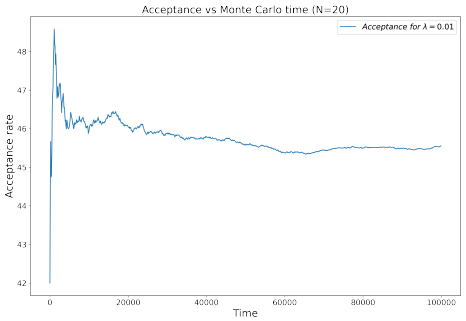
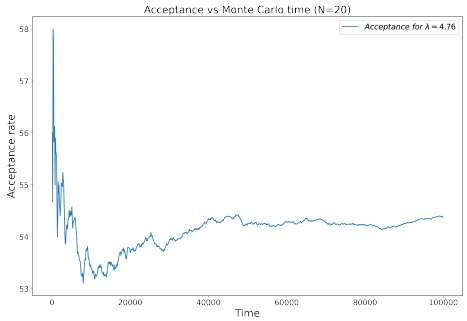


Figure 6: Polyakov loop values for the unitary matrix model in an external field.

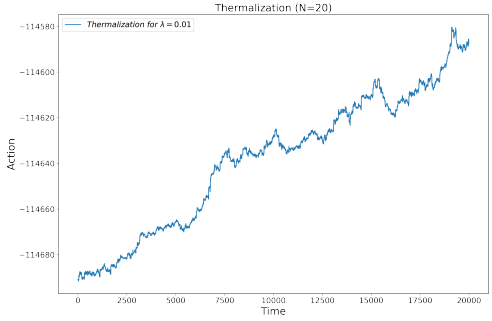


(a) Acceptance rate for $\lambda = 0.01$

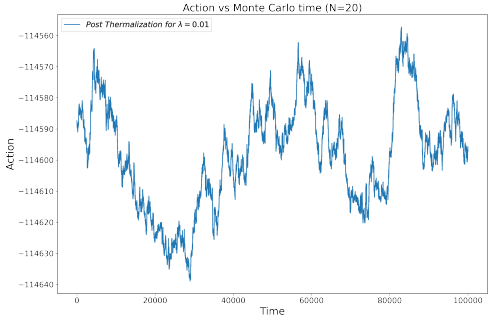


(b) Acceptance rate for $\lambda = 4.76$

Figure 7: Acceptance rate for the MCMC simulation of the unitary matrix model in an external field.

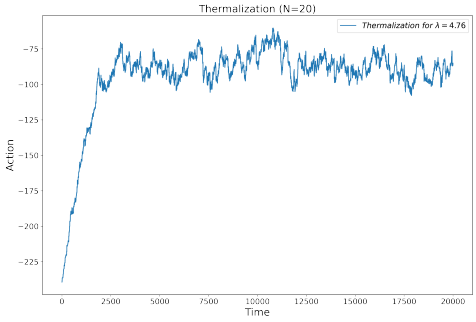


(a) Thermalization for $\lambda = 0.01$

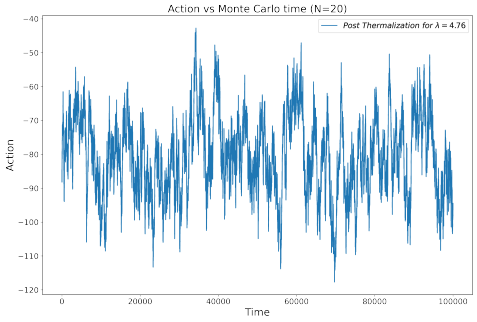


(b) Action for $\lambda = 0.01$

Figure 8: Thermalization and action for the MCMC simulation of the unitary matrix model in an external field for $\lambda = 0.01$.



(a) Thermalization for $\lambda = 4.76$



(b) Action for $\lambda = 4.76$

Figure 9: Thermalization and action for the MCMC simulation of the unitary matrix model in an external field for $\lambda = 4.76$.

5.3.3 Comparison

From Fig.10 we can see that the Polyakov loop values for the deformed GWW deviates from that of the the GWW Model. But both of them seem to undergo the phase transition at $s = \lambda = 2$.

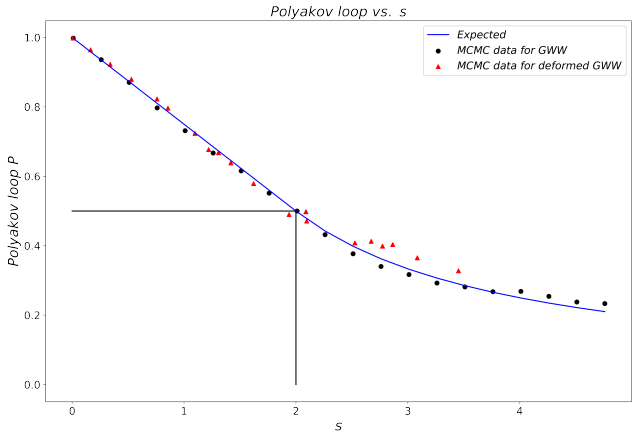


Figure 10: Comparison of the simulated values of the Polyakov loop for the GWW model and the unitary matrix model in an external field.

5.4 Windings of Polyakov Loop

We have also used the Polyakov loops with k windings to verify the theory. We have

$$P_k = \frac{1}{N}\langle \mathrm{Tr} U^k \rangle. \tag{34}$$

For the GWW Model we have

$$P_2 = \begin{cases} (1-\frac{\lambda}{2})^2 & \text{for} \quad \lambda < 2 \\ 0 & for \quad \lambda \geq 2 \end{cases} \tag{35}$$

$$P_3 = \begin{cases} -(1-\frac{\lambda}{2})^2(1-\frac{5}{4}\lambda) & \text{for} \quad \lambda < 2 \\ 0 & \text{for} \quad \lambda \geq 2 \end{cases} \tag{36}$$

$$P_4 = \begin{cases} (1-\frac{\lambda}{2})^2(1-3\lambda+\frac{7}{4}\lambda+\cdots) & \text{for} \quad \lambda < 2 \\ 0 & \text{for} \quad \lambda \geq 2 \end{cases} \tag{37}$$

5.4.1 Second Winding of the Polyakov Loop

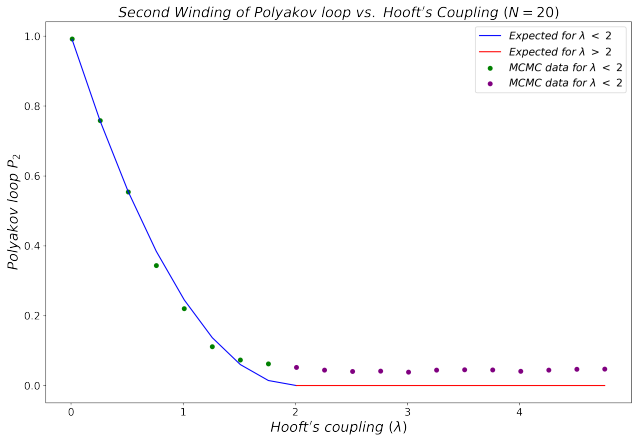
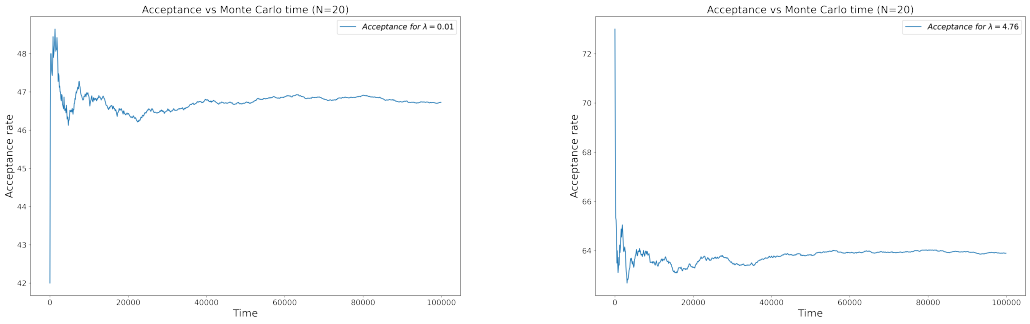
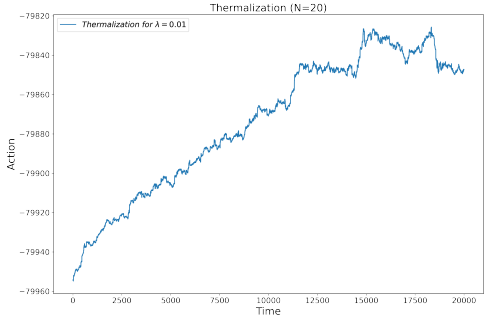


Figure 11: Comparison of the simulated values of the Polyakov loop with analytical values for the GWW Model.

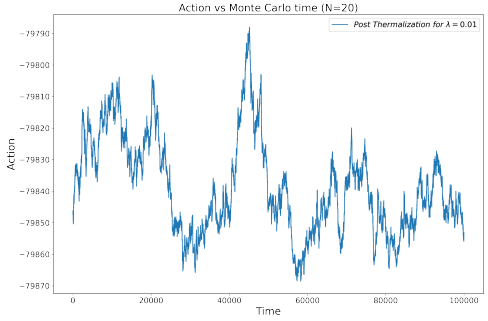


(a) Acceptance rate for $\lambda = 0.01$ (b) Acceptance rate for $\lambda = 4.76$

Figure 12: Acceptance rate for the MCMC simulation of the GWW model.

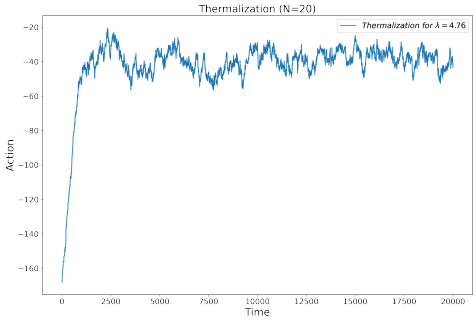


(a) Thermalization for $\lambda = 0.01$

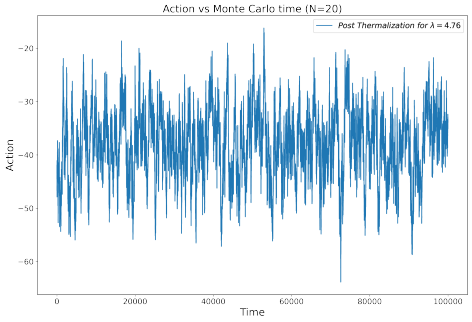


(b) Action for $\lambda = 0.01$

Figure 13: Thermalization and action for the MCMC simulation of the GWW model for $\lambda = 0.01$.



(a) Thermalization for $\lambda = 4.76$



(b) Action for $\lambda = 4.76$

Figure 14: Thermalization and action for the MCMC simulation of the GWW model for $\lambda = 4.76$.

5.4.2 Third Winding of the Polyakov Loop

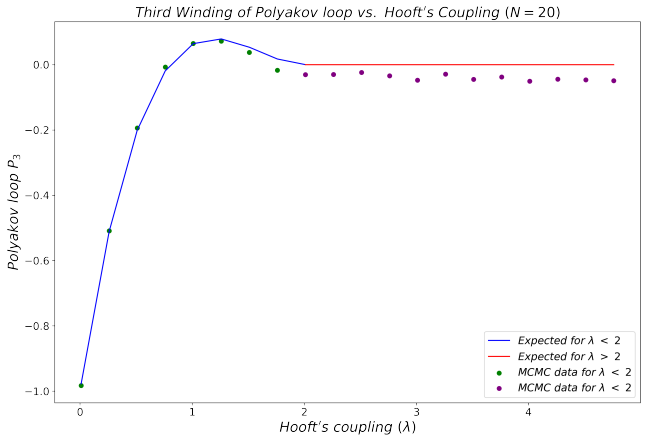
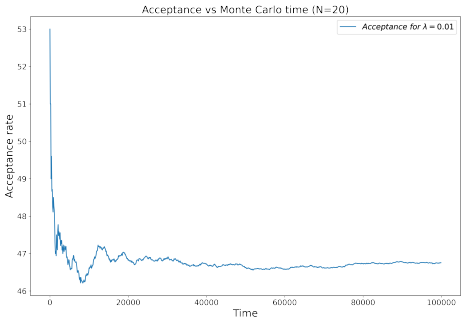
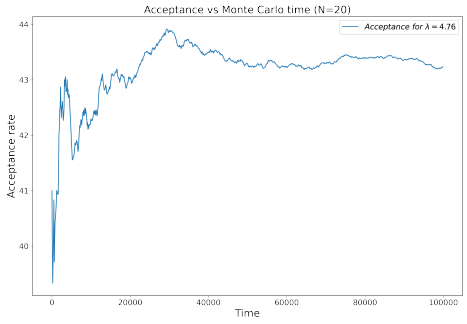


Figure 15: Comparison of the simulated values of the Polyakov loop with analytical values for the GWW Model.

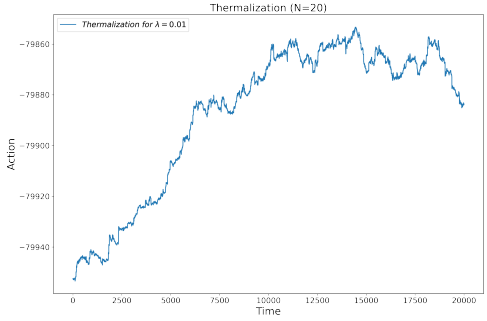


(a) Acceptance rate for $\lambda = 0.01$

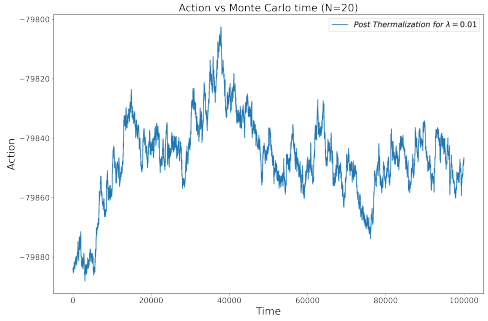


(b) Acceptance rate for $\lambda = 4.76$

Figure 16: Acceptance rate for the MCMC simulation of the GWW Model.

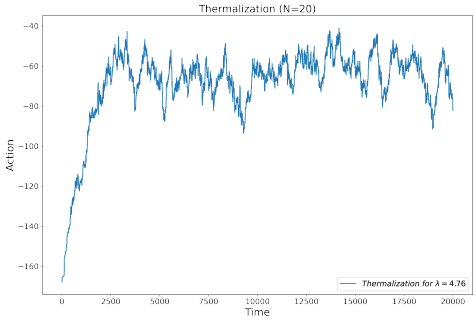


(a) Thermalization for $\lambda = 0.01$

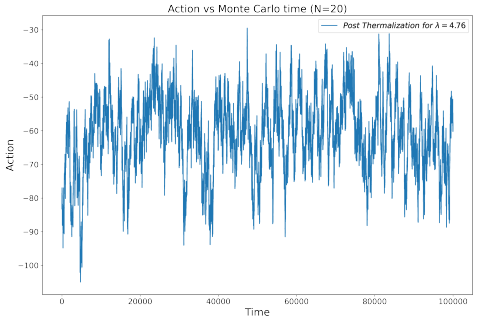


(b) Action for $\lambda = 0.01$

Figure 17: Thermalization and action for the MCMC simulation of the GWW model for $\lambda = 0.01$.



(a) Thermalization for $\lambda = 4.76$



(b) Action for $\lambda = 4.76$

Figure 18: Thermalization and action for the MCMC simulation of the GWW model for $\lambda = 4.76$.

5.4.3 Fourth Winding of Polyakov Loop

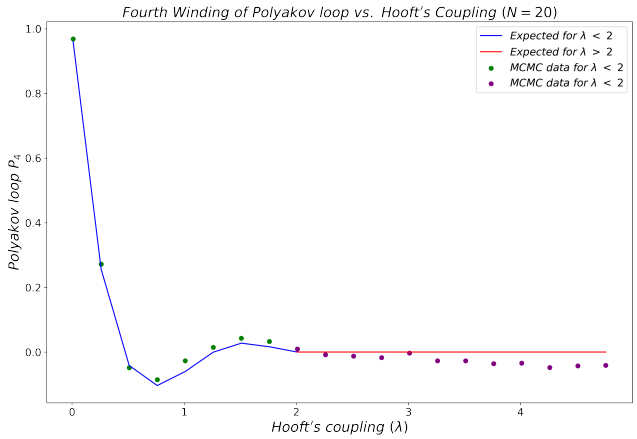
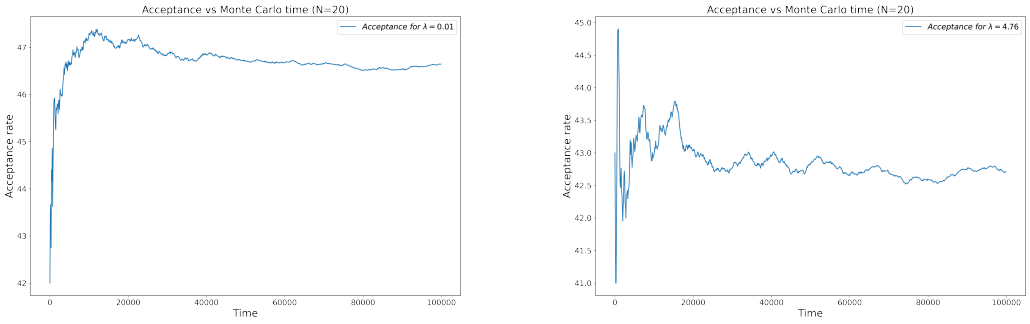
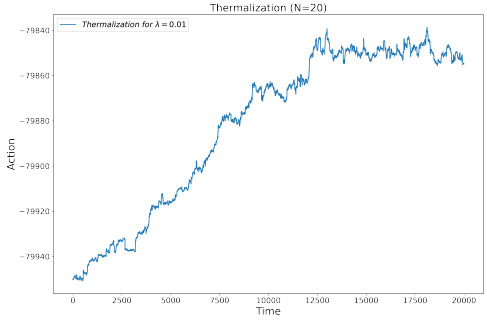


Figure 19: Comparison of the simulated values of the Polyakov loop with analytical values for the GWW Model.

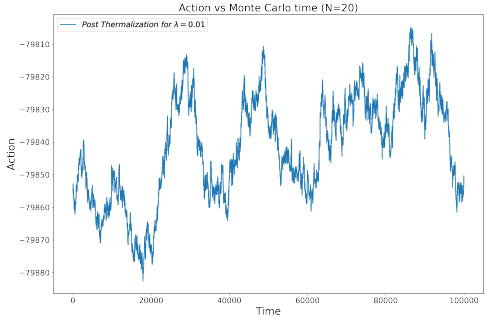


(a) Acceptance rate for $\lambda = 0.01$ (b) Acceptance rate for $\lambda = 4.76$

Figure 20: Acceptance rate for the MCMC simulation of the GWW Model.

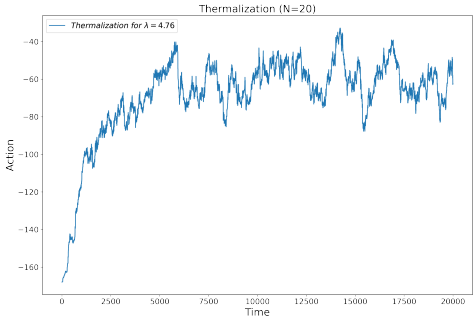


(a) Thermalization for $\lambda = 0.01$

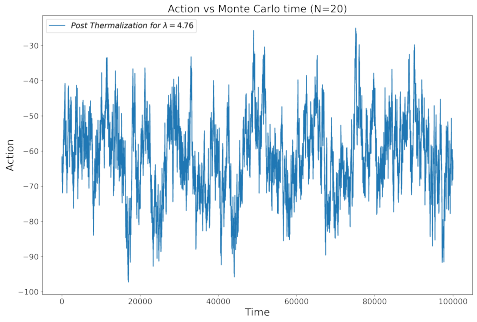


(b) Action for $\lambda = 0.01$

Figure 21: Thermalization and action for the MCMC simulation of the GWW model for $\lambda = 0.01$.



(a) Thermalization for $\lambda = 4.76$



(b) Action for $\lambda = 4.76$

Figure 22: Thermalization and action for the MCMC simulation of the GWW model for $\lambda = 4.76$.

6 Conclusions

For the GWW model the analytical solutions to the Polyakov loop observable were in close agreement with that simulated using Metropolis algorithm. The higher windings of the Polyakov loop also showed decent agreement between analytical and simulated values. To ensure the validity of the simulated values it was checked that action was thermalized and acceptance rate was within a reasonable range.

References

[1] E. Brezin and D. J. Gross, "*The External Field Problem in the Large N Limit of QCD*," Phys. Lett. B **97**, 120-124 (1980) doi:10.1016/0370-2693(80)90562-6

[2] D. J. Gross and M. J. Newman, "*Unitary and Hermitian matrices in an external field*," Phys. Lett. B **266**, 291-297 (1991) doi:10.1016/0370-2693(91)91042-T

[3] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller and Edward Teller, "*Equation of State Calculations by Fast Computing Machines*," J. Chem. Phys. **21**, 1087 (1953) https://doi.org/10.1063/1.1699114

[4] Anosh Joseph, "*Markov Chain Monte Carlo Methods in Quantum Field Theories*," SpringerBriefs in Physics **ISBN**: 9783030460440 **ISSN**: 2191-5431 (2020) doi:10.1007/978-3-030-46044-0 [arXiv:1912.10997v3 [hep-th]]

[5] Joshua S. Speagle, "*A Conceptual Introduction to Markov Chain Monte Carlo Methods*," (2020) [arXiv:1909.12313 [stat.OT]]

[6] Qingkai Kong Timmy Siauw Alexandre Bayen, "*Python Programming and Numerical Methods*," Academic Press (2020) **Paperback ISBN**: 9780128195499 **eBook ISBN**: 9780128195505