Introduction to Monte Carlo methods for matrix models

Raghav G. Jha[a]

Perimeter Institute for Theoretical Physics, Waterloo, Ontario N2L 2Y5, Canada

raghav.govind.jha@gmail.com

Abstract: Matrix models play an important role in Physics ranging from nuclear physics where they were first introduced to study of random surfaces, conformal field theories, integrable systems, two-dimensional quantum gravity, and non-perturbative descriptions of M-theory. In these notes, we consider a wide variety of matrix models and study them using Monte Carlo methods in the large N limit. The agreement with exact analytical results and recent bootstrap results are shown and expectations for some unsolved models are provided to assist future bootstrapping. In order to encourage this exchange, we provide codes in Python for convenience of the reader and explain how it can be modified to study other potentials. We hope these notes will be useful for further numerical/analytical investigations of matrix models for those who are not familiar with Monte Carlo methods. The codes we present were tested on a laptop and took between few minutes to couple of hours to finish.

Last edited by RGJ: 2021-10-17 at 11:18:43

Contents

SECTION 1

Introduction

Most of the phenomenon occurring in nature can be described by aa system which may or may not be solvable. In this pursuit, from very early days, matrices have played an important role in Physics ranging from development in quantum mechanics to understanding the large scale structures of the universe. On the other hand, several physical systems are known to sometimes be determined to a great extent by normally distributed elements (Gaussian distribution). It is the most important probability distribution in statistics because it fits many natural phenomena. It is appropriate here to quote a statement by Mark Kac [1] - "That we are led here to the normal law (distribution), usually associated with random phenomena, is perhaps an indication that the deterministic and probabilistic points of view are not as irreconcilable as they may appear at first sight". The subject of random matrix theory is the study of matrices whose entries are random variables chosen from a well-defined distributions. One of the remarkable things that was first noticed by Wishart around 1928 is that one can consider a family of probability distributions which is defined over symmetric, nonnegative-definite random matrices sometimes also known as matrix-valued random variables and this is now known by 'Wishart ensembles'. These are sometimes also known as 'Wishart-Laguerre' because the spectral properties of this distribution involve use of Laguerre polynomials.

The application of random matrices to physical problems was not until the 1950s when it took the genius of Wigner to first apply the random matrix theory in understanding the energy spectrum in nuclei of heavy elements. It was experimentally shown that unlike the case when the energy levels are assumed to be uncorrelated random numbers and the variable s would be governed by the familiar Poisson distribution i.e., $P(s) = e^{-s}$, there was more to this story and the distribution was nothing like Poisson. He realized (what is now known by the name 'Wigner's surmise'[2]) that it could be described by a distribution given by $P(s) = \pi s/2\, e^{-\pi s^2/4}$. The linear growth of $P(s)$ for small s is due to quantum mechanical level repulsion which was first considered by von Neumann and Wigner in 1929. This surmise and the paper Wigner had written in 1951 [1] introduced random matrix theory to nuclear physics and then in later decades to almost all of Physics.

This program was further continued in the 1960s, when in their exploration of random matrices, Dyson and Mehta studied and classified three types (sometimes also called 'the threefold way') of matrix ensembles with different correlations. The first was 'Gaussian orthogonal ensemble' which was used to describe systems with time reversal invariance and

[1] Kac was a Polish American mathematician. His main interest was probability theory. He is also known apart from other things for his thought provoking question - "Can one hear the shape of a drum?"

[2] Why is this called a 'surmise'? As is noted in the literature, the story goes like this: At some conference on Neutron Physics at the Oak Ridge National Laboratory in 1956, someone in the audience asked a question about the possible shape of the distribution of the spacings of energy levels in a heavy nucleus. Wigner who was in the audience, walked up to the blackboard and guessed the answer given above.

integer spin with weakest level repulsion between neighboring levels and had $\beta = 1$. The second was the Gaussian unitary ensemble with no time reversal invariance with intermediate level repulsion and $\beta = 2$. The third was the Gaussian symplectic ensemble for time reversal invariance for half integer spin with $\beta = 4$. These are now known as GOE, GUE, and GSE respectively [3]. The general $P(s)$ is given by, $c_\beta s^\beta e^{-a_\beta s^2}$ where $\beta \in (1, 2, 4)$ depending on the symmetry in question. For example, the nuclear data for heavy elements nearest neighbour spacing distribution is closely related to that of GOE distribution, see Fig. 2. Dyson's and Mehta's work made it more precise and improved further compared to what is shown in Fig. 2 as explained in [2]. We mention c_β and a_β in Table (1). We show the three distributions using MATHEMATICA in Fig. 1.

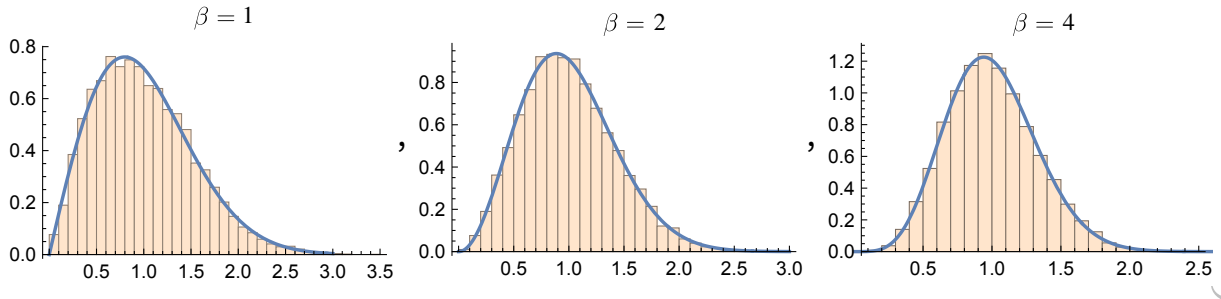

Figure 1. The distribution of three ensembles mentioned in the text.

β	c_β	a_β
1	$\pi s/2$	$\pi/4$
2	$32s^2/\pi^2$	$4/\pi$
4	$2^{18}s^4/3^6\pi^3$	$64/9\pi$

Table 1. We mention the values for c_β and a_β for different ensembles.

It was later concluded that much to Wigner's own surprise, his guess was fairly accurate as shown and improved by Mehta [3] and Gaudin [4]. Apart from its extensive use in Physics, it is now clear that the field of random matrix theory is intimately related to number theory (especially the pair correlation of the zeros of the Riemann-zeta function) and this was observed by Montgomery and Dyson at a tea break at the Institute for Advanced Study in the 1970s. Some still believe that any future proof of Riemann hypothesis lies in the deepest secrets of random matrix theory. This belief is also the theme of the idea proposed independently by Hilbert and Pólya who suggested that the zeros of the zeta function might be the

[3]For example, GUE represents a statistical distribution over complex Hermitian matrices have probability densities proportional to $\exp(-\mathrm{Tr}(A^2/2\sigma^2))$ and where matrix elements i.e., a_{ij} are an independent collection of complex variates whose real and imaginary parts are from normal distribution with zero mean ad unit variance. In MATHEMATICA , we can use: `GaussianUnitaryMatrixDistribution[σ,N]` to get a $N \times N$ such matrix

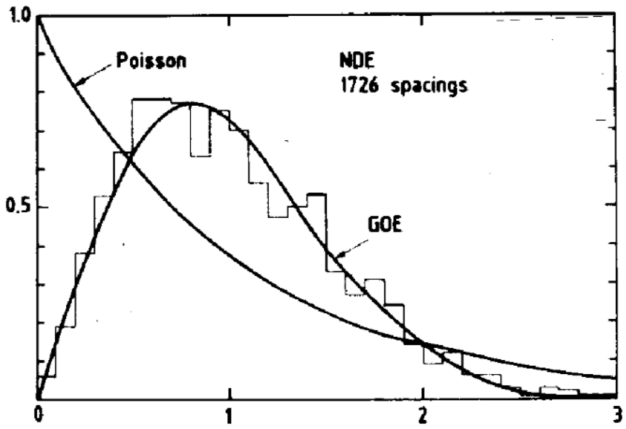

Figure 2. The nearest neighbour spacing distribution (i.e., $P(s)$) for nuclear data. The GOE (Gaussian Orthoghonal Ensemble) and Poisson is given by solid curves. This figure is taken from - "Fluctuation Properties of Nuclear Energy Levels and Widths : Comparison of Theory with Experiment" by O. Bohigas, R. U. Haq, and A. Pandey

eigenvalues of some unknown Hermitian 'operator'. It is another thing that no one yet knows such an operator! We refer the reader to the excellent books [5, 6] for introductions to the field of random matrix theory.

As far as Physics and quantum field theories is concerned, the major development with the matrix degrees of freedom and matrix models came with the work of 't Hooft in 1974. By then, it was almost clear that the correct theory of strong interactions is QCD where there are matrix degrees of freedom for gauge fields based on $SU(3)$ gauge group. 't Hooft proposed to consider a general $SU(N)$ symmetry since he showed that in such a limit only planar diagrams survive and calculations becomes more tractable. In fact, this large N limit was later understood to be closely related to some description of string theory. This work brought together the idea of random matrix models in the asymptotic limit (large matrices) to the mainstream Physics and is extremely fruitful till date. This also enabled us to study several interesting features of quantum gravity from a field theoretic point of view through the famous AdS/CFT conjecture.

There are some excellent reviews about random matrix theory, matrix integrals/models and their formal aspects which we leave out for this version of the article. We refer the reader to two excellent reviews written 23 years apart [7, 8] for detailed discussions. The goal of these notes is to introduce the numerical solutions of matrix models using Monte Carlo methods and use it to solve many models where no other treatment is possible. The motivation for this stems from the recent progress in numerical bootstrap program to solve matrix models and we hope that these notes with the PYTHON codes will supplement those explorations and serve as a cross-check of the results. All the codes in these notes can be accessed at:

https://github.com/rgjha/MMMC

The plan of the article is as follows. In Sec. 1, we mention saddle-point solution to one matrix model and give a brief discussion about the orthogonal polynomials and how it can be used to solve a unitary matrix model and two matrix model description of Ising model on random graph. Then in the early part of Sec.2, we briefly discuss the numerical bootstrap methods and then concentrate for bulk of remaining section on explaining the basics of MC method and use it to solve various models. We end the notes with a hope on how the close association between MC and bootstrap can serve as a useful guide to solve matrix models.

SECTION 2

Matrix models - Analytical results

2.1 One-plaquette model - Orthogonal Polynomials

In these matrix models, one often needs to evaluate integrals of the form:

$$Z = \int \exp\Big[- \mathrm{Tr} \sum V(M_i) - \mathrm{Tr} \sum c_{ij} M_i M_j \Big] dM_i dM_j \tag{2.1}$$

where M^i are $N \times N$ complex Hermitian matrices.

In study of zero-dimensional gauge theories, we encounter these types of matrix models where the integral is over the Haar measure. These models often have interesting features in the large N limit. For ex:, in QCD, the gauge fields are elements of SU(3) and hence they are well-described in some cases by unitary matrix models. A general model is given by:

$$Z = \int \exp\Big[- \mathrm{Tr} \sum V(U_i) - \mathrm{Tr} \sum c_{ij} U_i U_j \Big] dU_i dU_j \tag{2.2}$$

where U^i are $N \times N$ unitary matrices.

One of the well-known examples is the unitary matrix model first considered by Gross, Witten, and Wadia (GWW) around 1979. This is often known as one-plaquette matrix model. This unitary matrix model has deep connections to string theory in the so called 'double-scaling limit' where $N \to \infty$ and $\lambda \to \lambda_c$ simultaneously. The requirement of this double scaling can be understood as follows: If we merely take $N \to \infty$ then we get genus zero surfaces in the expansion of the free energy. However, this would prohibit string interaction since they would imply change of genus which is not possible in that limit. In taking DSL, this problem is resolved and topological information is maintained. This model is given by :

$$Z = \int \exp\Big[- \mathrm{Tr}(U + U^\dagger) \Big] dU \tag{2.3}$$

This model admits exact solution for all N and λ in terms of determinant of a Toeplitz matrix. However, it is not very useful and hence this model has been studied by saddle-point methods and orthogonal polynomials (OP). We will sketch the solution using OP closely following [9]. Following the discussion in the Appendix, we define the polynomial:

$$P_j(x) = \sum_{k=0}^{j-1} b_k x^k + x^j. \tag{2.4}$$

We have to choose polynomials which are orthonormal with respect to the measure which in this case is:

$$\rho(\theta) = \exp\Big[\frac{2N}{\lambda}\cos\theta\Big], \tag{2.5}$$

and this results in:

$$\int_{-\pi}^{\pi} \rho(\theta) P_m(e^{i\theta}) P_n^*(e^{i\theta}) = a_m \delta_{mn} \tag{2.6}$$

Denoting $\kappa = 2N/\lambda$ then we have,

$$I_n(\kappa) = I_{-n}(\kappa) = \int_{-\pi}^{\pi} \rho(\theta) e^{in\theta} \tag{2.7}$$

and the polynomials defined above are given by:

$$P_n(z) = \det \begin{pmatrix} I_0 & I_1 & \cdots & I_n \\ I_1 & I_0 & \cdots & I_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & z & \cdots & z^n \end{pmatrix} \frac{1}{\det[I_{i-j}]_{i,j=1\cdots N}} \tag{2.8}$$

We note that the coefficients of $P_n(z)$ are real and hence $P_n^*(e^{i\theta}) = P_n(e^{-i\theta})$. It is also staright-forward by expanding the determinant to see that: $a_n = c_{n+1}/c_n$ where $c_N = \det[I_{i-j}]_{i,j=1\cdots n}$ is the Toeplitz determinant. In fact, it is easy to show that for U(N) model:

$$Z = \det\Big[I_{i-j}(\kappa)\Big] \tag{2.9}$$

• Exercise 1: How does the exact result (2.9) change when N is finite and we consider $SU(N)$ rather than $U(N)$ gauge group.

2.2 One-matrix model with cubic and quartic potentials - Saddle point analysis

Before we delve into the details of how to solve the one matrix model using saddle point (or 'stationary phase') method, we discuss the basics of this in a simple setting where we deal

with integrals over a large number N of variables. Suppose we want to evaluate the integral given by:

$$I(\alpha) = \lim_{\alpha \to 0} \int_{-\infty}^{\infty} e^{-\frac{1}{\alpha} f(x)} \, dx, \tag{2.10}$$

where α is a positive integer but we would like to eventually consider the interesting limit of $\alpha \to 0$ and f(x) is a real valued function. In the limit where α becomes small, the exponential causes the integrand to peak sharply at function's minima. There might be several extrema, But the integral will be dominated by one which minimizes $f(x)$ as $\alpha \to 0$ (let it be x_0), we use the Taylor expansion about the saddle point x_0 and throw away higher-order terms to get:

$$f(x) = f(x_0) + f''(x_0)(x - x_0)^2 + \cdots \tag{2.11}$$

Using (2.11) in (2.12) along with the famous Gaussian integral result i.e., $\int_{-\infty}^{\infty} e^{-\alpha x^2} dx = \sqrt{\frac{\pi}{\alpha}}$ we get the desired result:

$$I(\alpha) = \sqrt{\frac{2\pi\alpha}{f''(x_0)}} e^{-f(x_0)/\alpha}. \tag{2.12}$$

• Exercise 2: Find the terms which are of $\mathcal{O}(\alpha)$ in (2.10) by doing little more work and show that:

$$I(\alpha) = \int_{-\infty}^{\infty} e^{-\frac{1}{\alpha} f(x)} \, dx = \sqrt{\frac{2\pi\alpha}{f''(x_0)}} e^{-f(x_0)/\alpha} \left[1 + \left[\frac{5}{24} \frac{(f''')^2}{(f'')^3} - \frac{3}{24} \frac{f''''}{(f'')^2} \right] \alpha + \mathcal{O}(\alpha^2) \right].$$

The one matrix hermitian model using saddle point method was famously solved by Brezin-Itzykson-Parisi-Zuber (BIPZ) [10]. This solution is standard and can be found in any standard review or textbooks [7, 11]. We will briefly sketch the solution for sake of completeness. This model is solved using the method of resolvent as described below:

$$Z = \int dM \exp\Big[-\frac{N}{g} \mathrm{Tr} V(M) \Big] \tag{2.13}$$

$$= \int \prod d\lambda_i \Delta^2(\lambda) e^{-\frac{N}{g} \sum V(\lambda_i)} \tag{2.14}$$

where $\Delta(\lambda) = \prod_{i>j} (\lambda_i - \lambda_j) = \exp\Big[\sum_{i>j} \log|\lambda_i - \lambda_j| \Big]$ is the Vandermonde determinant. If we vary one of the eigenvalues, it gives the saddle point equation:

$$\frac{2}{N} \sum_{j \neq i} \frac{1}{\lambda_i - \lambda_j} = \frac{1}{g} V'(\lambda_i). \tag{2.15}$$

It is useful to introduce the density of eigenvalues as:

$$\rho(\lambda) = \frac{1}{N}\sum_{i=1}^{N}\delta(\lambda - \lambda_i). \tag{2.16}$$

In the limit of large N, we can write (2.15) as:

$$V'(\lambda) = 2g \fint_b^a d\mu \frac{\rho(\mu)}{\lambda - \mu} \tag{2.17}$$

where by $\fint$ we meant the Cauchy principal value of the integral. For often deal with symmetric single-cut such that $b = -a$. We can write resolvent by noting that it is the Stieljes transform of the eigenvalue density as:

$$G(z) = 2\fint_{-a}^{a} d\mu \frac{\rho(\mu)}{z - \mu} \tag{2.18}$$

which we can write then using Sokhotski-Plemelj theorem as:

$$G(z \pm i\epsilon) = \fint_{-a}^{a} d\mu \frac{\rho(\mu)}{z - \mu} \mp i\pi\rho(z), \tag{2.19}$$

which is equivalent using 2.17 to:

$$G(z \pm i\epsilon) = \frac{1}{2g}V'(z) \mp i\pi\rho(z). \tag{2.20}$$

Once we find the resolvent, we can solve the model and find the moments. it is useful to mention here that we can use the useful closed expression for resolvent in terms of a contour integral (see for example (A.24) of [12]) as given by:

$$G(x) = \int_{-a}^{a} \frac{1}{2\pi i}\frac{\sqrt{x^2 - a^2}}{\sqrt{y^2 - a^2}} NV'(y)\frac{1}{x - y}, \tag{2.21}$$

for symmetric cuts and by,

$$G(x) = \oint \frac{dy}{2\pi i}\sqrt{\frac{(x-a)(x-b)}{(y-a)(y-b)}} NV'(y)\frac{1}{x - y}, \tag{2.22}$$

if the cut was instead $C = [b, a]$. For the case of 1MM with quartic potential i.e., $V(M) = M^2/2 + gM^4/4$, one obtains the exact result (for $g \geq -1/12$):

$$t_2 = \frac{(12g + 1)^{3/2} - 18g - 1}{54g^2}. \tag{2.23}$$

The MATHEMATICA code to solve this model is given in the Appendix.

- Exercise 3: Show that $\det(V) = \prod_{i<j}(\lambda_i - \lambda_j)$ where V is given by:

$$V = \begin{pmatrix} 1 & \lambda_1 & \lambda_1^2 & \cdots & \lambda_1^{N-1} \\ 1 & \lambda_2 & \lambda_2^2 & \cdots & \lambda_2^{N-1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & \lambda_N & \lambda_N^2 & \cdots & \lambda_N^{N-1} \end{pmatrix} = \lambda_i^{j-1}$$

- Exercise 4: Derive the loop equations (aka Schwinger-Dyson equations) given below:

$$\Big\langle \mathrm{Tr} M^k V'(M) \Big\rangle = \sum_{l=0}^{k-1} \langle \mathrm{Tr} M^l \rangle \langle \mathrm{Tr} M^{k-l-1} \rangle$$

2.3 Open p-matrix chain model

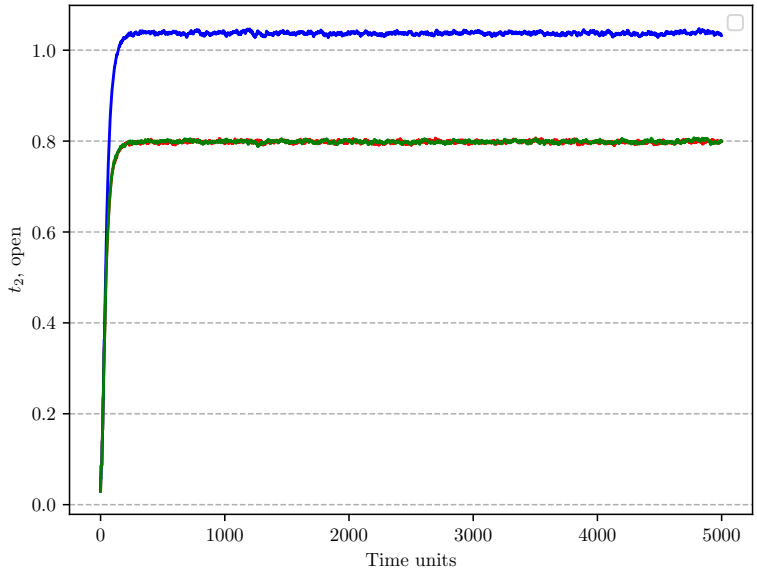

Figure 3. We see that two matrices have common $t_2 = 0.798(3)$ and the third has $t_2 = 1.037(3)$. This is for $g = 1$ for $N = 300$. We have computed errors after discarding first 1000 time units and using jackknife blocking.

The general p-matrix model chain was first considered and solved (for open case) in [13]. We will not their rather involved computation but instead show the results we obtain from

numerics in Fig. 3.

$$Z_p(g,c,\kappa) = \int \mathcal{D}M_1 \cdots \mathcal{D}M_p \exp \mathrm{Tr}\left(\sum_{i=1}^{p} -M_i^2 - gM_i^4 + c\sum_{i=1}^{p-1} M_i M_{i+1} + \kappa M_p M_1 \right). \quad (2.24)$$

This model *****

2.4 Matrix with external field

Gross Brezin paper...

SECTION 3

Numerical solutions

There is only a selected list of analytically solvable matrix models in the planar limit. This inevitably brings the thought of attempting numerical solutions. Unfortunately, even here, there are not many methods which one can use. In fact, there only comes two methods to mind with the second method barely few years old. This clearly signals the fact that there still remains lot of work to be done in devising new numerical methods to solve matrix models. The most frequently used method is Monte Carlo (MC)[4]

is quite effective but is not a panacea. The newly introduced numerical bootstrap method has already been used to solve several matrix models [14–17] but the extension to matrix models with more than two matrices i.e., 3-matrix commutator type models or IKKT models seems difficult at the moment. This is related to the fact that with more matrices, the loop equations which become highly non-linear are difficult to handle. In these notes, we will mostly focus on MC methods since as of this writing bootstrap methods have not been able to access or constrain matrix models with more than two matrices. However, we want to explain the bootstrap solution for the case of one matrix model as first shown by [15] for the interested reader. For detailed exposition, we refer the reader to the references above. We hope that this 'yet' small section on bootstrap methods would be extended in later versions of these notes as more results accumulate in coming years.

3.1 Numerical bootstrap

The basic idea of bootstrapping matrix models depends on the positivity (positive-definiteness) of the bootstrap matrix which we refer in what follows by $\mathcal{M}$. For the case of one matrix

[4] It is not widely known that Monte Carlo methods were central to the work required for the Manhattan Project and first introduced in the 1940s by Stanislaw Ulam and recognized first by Von Neumann. He is often credited as the inventor of modern version of the Markov Chain Monte Carlo method. In fact, the earliest use of MC goes back to the solution of Buffon needle problem when Fermi used it in 1930s but never published. Since this work was part of the classified Information, the work of von Neumann and Ulam required a code name. It was Metropolis who suggested using the name Monte Carlo based on the casino in Monaco where Ulam's uncle would borrow money from relatives to gamble. Ulam and Metropolis published first paper on MC in 1949 titled 'The Monte Carlo Method'.

model (1MM) with a potential $V(X)$ given by:

$$V(X) = \frac{1}{2}X^2 + \frac{g}{4}X^4, \tag{3.1}$$

it is straightforward to show that the odd moments of X i.e., $t_n = \text{Tr}X^n = 0$ for odd n while the even moments (of order greater than two) are all related to $t_2 = \text{Tr}X^2/N$. This renders the model simple to bootstrap since there is no growth of words (combination of matrix or matrices!) since all non-zero t_n can be related to t_2.

• Exercise 5: Use loop equations and show that for the 1MM with quartic potential, it is possible to write t_4, t_6, t_8 in terms of t_2. Also check this using either MATHEMATICA or PYTHON [see Appendix for codes]. Repeat this exercise for cubic potential where higher moments can be written in terms of t_1.

If we consider positive constraints that can be derived from $\langle \text{Tr}(\Phi^\dagger \Phi)\rangle \geq 0$ where Φ is superposition of matrices which for one matrix model is $\Phi = \sum_k \alpha_k M^k$. This condition is equivalent to positive definite nature of $\mathcal{M} \succeq 0$ where $\mathcal{M}_{ij} = \langle \text{Tr}M^{i+j}\rangle$. We can only enforce subset of these constraints. For example, it was sufficient to access positive definite nature of $\mathcal{M}_{6\times 6} \succeq 0$ and sub-matrices to get to the exact solution in the original article [15]. For example, $\mathcal{M}_{2\times 2}$ is given by:

$$\mathcal{M}_{jk} = \begin{pmatrix} t_{2j} & t_{j+k} \\ t_{j+k} & t_{2k} \end{pmatrix} \succeq 0 \tag{3.2}$$

In this case, solving the model just means finding the bounds on t_2 since all others can then be calculated (see the exercise above).

Following this work, a two matrix quantum mechanics model (in 0+1-dimensions) was solved using similar techniques. The results obtained were shown to be consistent with MC results. In this work, the Hamiltonian considered was given by:

$$H = \text{Tr}\Big(P^2 + X^2 + \frac{g}{N}X^4\Big) \tag{3.3}$$

and corresponding to the trial operators up to length (L) = 2, they considered $\mathbb{I}, X, X^2$ and P. The bootstrap matrix of size $2^L \times 2^L$ which should be positive definite is constructed as:

$$\mathcal{M} = \begin{pmatrix} \langle \text{Tr}\mathbb{I}\rangle & \langle \text{Tr}X^2\rangle & 0 & 0 \\ \langle \text{Tr}X^2\rangle & \langle \text{Tr}X^4\rangle & 0 & 0 \\ 0 & 0 & \langle \text{Tr}X^2\rangle & \langle \text{Tr}XP\rangle \\ 0 & 0 & \langle \text{Tr}PX\rangle & \langle \text{Tr}P^2\rangle \end{pmatrix} \succeq 0 \tag{3.4}$$

They considered bootstrap matrices up to $L = 4$ and observed convergence to the expected result. For the case of two-matrix quantum mechanics, the convergence was found to be

slow but consistent with results expected using Monte Carlo results and bounds from Born-Oppenheimer wavefunction.

Another two-matrix integral model given by the action (3.20) was recently solved [17]. In this work, authors used relaxation bootstrap methods for $\Lambda = 11$ (determining the length of largest words considered) to constrain the moment to five decimal places. They obtained $\mathrm{Tr}X^2/N = t_2$ to be bounded between $0.421783612 \leq t_2 \leq 0.421784687$ while $\mathrm{Tr}X^4/N = t_4$ between $0.333341358 \leq t_4 \leq 0.333342131$ for the symmetric phase (i.e.,) We will come back to discuss this model for both symmetric and symmetry broken phase and its corresponding solution using MC methods in **SECTIO —**

3.2 Monte Carlo method

The numerical method which is state-of-the-art in classical computations of matrix models and quantum field theories in general is the Monte Carlo approach. For higher-dimensional models, one starts with the lattice formulation which reduces the path-integral introduced by Feynman into many ordinary integrals. But even for a simple gauge theory like $\mathbb{Z}_2$ in four dimensions this is not practical to evaluate. The fact that we need to do so many integrals suggests that may be some statistical interpretation and this is where the basic idea of Monte Carlo comes in. It selects the configuration which dominates the path integral and then ignores the rest of the available phase space.In that way one constructs a chain of configurations which can lead to the required distribution. Metropolis-Hastings algorithm and Hamiltonian/Hybrid Monte Carlo (HMC) are two frequently used methods which can generate a Markov chain and lead to unique stationary distribution. We will here focus on the latter since that has now become the state-of-the-art in various numerical computations. This method was introduced in 1987 by Duane, Kennedy, Pendleton, and Roweth who put together the ideas from Markov chain Monte Carlo (MCMC)[5] and molecular dynamics (MD) methods. For a detailed review about HMC and its extension to rational HMC which is required for fermions, the interested readers can consult the Refs. [18]. The two basic parts of HMC are: a) Use of integrator to evolve and propose a new configuration, b) accept or reject the proposed configuration. But before we discuss HMC, it is important to see how we generate random momentum matrices at start of each trajectory (time unit) since this is necessary to ensure that we converge to the correct answer using Monte Carlo methods.

3.2.1 Random number generator

It is essential during a Monte Carlo process that we correctly generate momentum matrices at the start of the leapfrog method taken from a Gaussian distribution. In this part, we will sketch this process and provide the accompanying code. Suppose we have two numbers U

[5]MCMC originated in the seminal paper of Metropolis et al **cite**, where it was used to simulate the state distribution for a system of ideal molecules.

and V taken from uniform distribution i.e., (0,1) and we want two random numbers with probability density function $p(X)$ and $p(Y)$ given by:

$$p(X) = \frac{1}{\sqrt{2\pi}} e^{-X^2/2} \tag{3.5}$$

and,

$$p(Y) = \frac{1}{\sqrt{2\pi}} e^{-Y^2/2} \tag{3.6}$$

Since X and Y are independent sets:

$$p(X,Y) = p(X)p(Y) = \frac{1}{2\pi} e^{-R^2/2} = p(R,\Theta) \tag{3.7}$$

where $R = X^2 + Y^2$. This then means that we can make the identification as given below:

$$U = \frac{\Theta}{2\pi}, \tag{3.8}$$

and,

$$V = e^{-R^2/2} \implies R = \sqrt{-2\log(V)}. \tag{3.9}$$

This then immediately means:

$$X = R\cos\Theta = \sqrt{-2\log(V)}\cos(2\pi U), \tag{3.10}$$

$$Y = R\sin\Theta = \sqrt{-2\log(V)}\sin(2\pi U). \tag{3.11}$$

The code to implement this in PYTHON is given below:

```
def box_muller():
# Basic form (alt: polar form) of Box-Muller random number generator for pairs of
# independent, standard, normally distributed (mean=0, sigma=1)
# given a source of uniformly distributed random numbers in (0,1)
# p, q have weights exp(-p^2/2)/sqrt(2pi) and exp(-q^2/2)/sqrt(2pi) respectively.

   PI = 2.0*math.asin(1.0);
   r = random.uniform(0,1)
   s = random.uniform(0,1)
   x = np.sqrt(-2.0*np.log(r)) * math.sin(2.0*PI*s)
   y = np.sqrt(-2.0*np.log(r)) * math.cos(2.0*PI*s)
   return x,y
```

It is easy to check that it indeed generates a Gaussian distribution with desired properties as shown in Fig. 4.

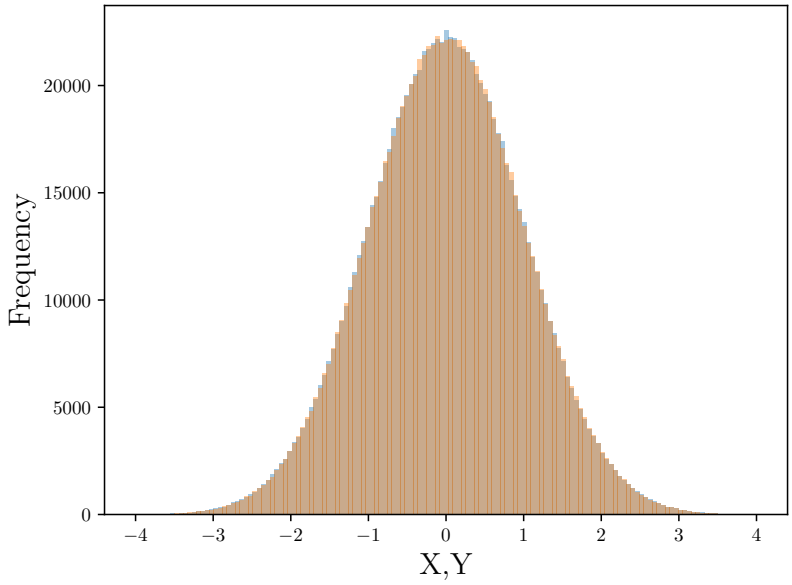

Figure 4. In the limit of large sample size (here 1 million), it tends to Gaussian with mean zero and $\sigma = 1$.

3.2.2 Leapfrog integrator and Metropolis step

The leapfrog method is used to numerically integrate differential equations. This is a second-order method and the energy non-conservation depends on square of step-size used. This method is 'symplectic', i.e., it preserves the area of the phase space. We can intuitively understand this as follows: Consider a rectangular region of area dA as shown below in Fig. (**??**). The four corners at time t are denoted by $(x, p), (x + dx, p), (x, p + dp), (x + dx, p + dp)$. At some later time t', this will change to form corners of some quadrilateral as shown with area dA'. It is then the statement of Liouville's theorem[6] that the areas are equal, i.e., $dA = dA'$. Using this idea we can easily prove an important equality used to check various lattice computations employing symplectic algorithms left as an exercise for the reader. The basic steps of the leapfrog algorithm are as follows:

- $X_i(\frac{\Delta\tau}{2}) = X_i(0) + P_i(0)\frac{\Delta\tau}{2}$
- Now several inner steps where $n = 1 \cdots (N-1)$

  $$P_i(n\Delta\tau) = P((n-1)\Delta\tau) - f_i((n-\tfrac{1}{2})\Delta\tau)\Delta\tau$$
  $$X_i((n+1/2)\Delta\tau) = X_i\Big(\Big(N-\tfrac{1}{2}\Big)\Delta\tau\Big) + P_i(n\Delta\tau)\Delta\tau$$

[6] Liouville theorem is closely related to detailed balance condition which says that in equilibrium there is balance between any two pairs of states i.e., equal probability.

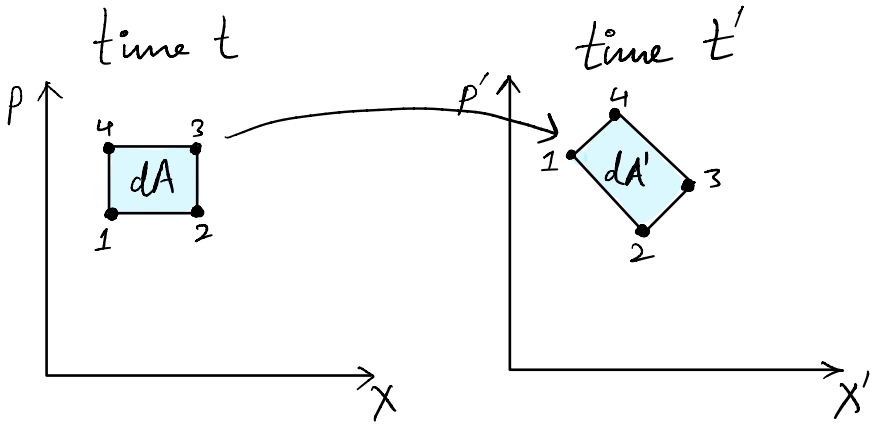

Figure 5. A representation of conservation of phase space area. This is also referred to as 'Liouville theorem. it asserts that under evolution of the system, it may change the shape of the shaded region but not the volume since probability has to be conserved.

- $P_i(N\Delta\tau) = P_i((N-1)\Delta\tau) - f_i((N-\frac{1}{2})\Delta\tau)\Delta\tau$
- $X_i(N\Delta\tau) = X_i\Big(\Big(N-\frac{1}{2}\Big)\Delta\tau\Big) + P_i(N\Delta\tau)\frac{\Delta\tau}{2}$

• Exercise 6: Show that a consequence of Liouville theorem is that $\langle e^{-\Delta H(\dot{)}}\rangle = 1$. Check this this also holds in any given HMC simulation within errors if a symplectic integrator is used.

In the last step of HMC, a Metropolis test is carried out to accept or reject the proposed configuration. Suppose we start from the configuration X of a one matrix model which is a $N \times N$ matrix and carry out the leapfrog part with some parameters and obtain a new configuration $X^\star$. The test then computes: $\texttt{min.}(1, e^{-\Delta H})$ and generates a uniform random number between $r \in [0, 1]$. The new configuration is rejected if $\texttt{min.}(1, e^{-\Delta H}) < r$ otherwise accepted.

3.2.3 Autocorrelation and jackknife errors

It must be kept in mind that given a Markov chain, the new states (i.e., configurations) can be highly correlated to previous ones. In order to ascertain that the measurement of an observable $\langle\mathcal{O}\rangle$ is not affected by correlated configurations, it is essential for proper statistical analysis to know the extent to which they are correlated. In this regard, it is important to measure the autocorrelation time τ_{auto} which measures the time it takes for two measurements to be considered independent of each other. So, if we generate L configurations, then

actually only $L/\tau_{\rm auto}$ are useful for computing averages. We define autocorrelation function of observable $\mathcal{O}$ as:

$$C(t) = \frac{\langle \mathcal{O}(t_0)\mathcal{O}(t_0+t)\rangle - \langle \mathcal{O}(t_0)\rangle\langle \mathcal{O}(t_0+t)\rangle}{\langle \mathcal{O}^2(t_0)\rangle - \langle \mathcal{O}(t_0)\rangle^2} \tag{3.12}$$

The behaviour of $C(t)$ is $\sim \exp(-t/\tau_{\rm auto})$ as $t \to \infty$. This is called exponential autocorrelation time. We can also compute something which is called 'integrated autocorrelation time' defined as:

$$\tau_{\rm auto}^{\rm int.} = \frac{\sum_{t=1}^{\infty}\langle \mathcal{O}(t_0)\mathcal{O}(t_0+t)\rangle - \langle \mathcal{O}\rangle^2}{\langle \mathcal{O}^2\rangle - \langle \mathcal{O}\rangle^2} \tag{3.13}$$

We can write this in terms of sum over autocorrelation function as: $\tau_{\rm auto}^{\rm int.} = 1 + \sum_{t=1}^{N} C(t)$. In general, $\tau_{\rm auto}$ increases with system size, close to critical point. Once can express the statistical error in the average of $\mathcal{O}$ denoted by $\delta\mathcal{O}$ is given in terms of variance and integrated autocorrelation time as:

$$\delta\mathcal{O} = \frac{\sigma}{\sqrt{N}}\sqrt{2\tau_{\rm auto}^{\rm int.}} \tag{3.14}$$

where we have usual definitions i.e., $\sigma = \sqrt{\langle \mathcal{O}^2\rangle - \langle \mathcal{O}\rangle^2}$ and N is the number of measurements.

3.3 One-matrix model with quartic & cubic potentials: Confirming exact results

With a quick introduction of the basic elements of the method, we want to use it to study some matrix model. For this purpose, the exact solution for one matrix model is a good testbed to check numerical results and compare to other different approaches. As we already mentioned in the section before, this has an exact solution and any moment, t_n can be obtained using the MATHEMATICA code given in the appendix. We show that the exact result, bootstrap result, and the Monte Carlo results all agree for $g = 1$ in Fig. 6. The PYTHON code which was used to produce this Figure can be found in the Appendix for the convenience of the reader. We also discuss how to run the code on your laptop and estimated time for completion. This code[7] is also available at:

https://github.com/rgjha/MMMC

We can also consider one-matrix model with cubic interaction instead of quartic as above. There is well-known instability associated with cubic potential. The model is given by:

$$V(M) = \frac{M^2}{2} + \frac{gM^3}{3} \tag{3.15}$$

[7]Please email author for any bug report or additional requests

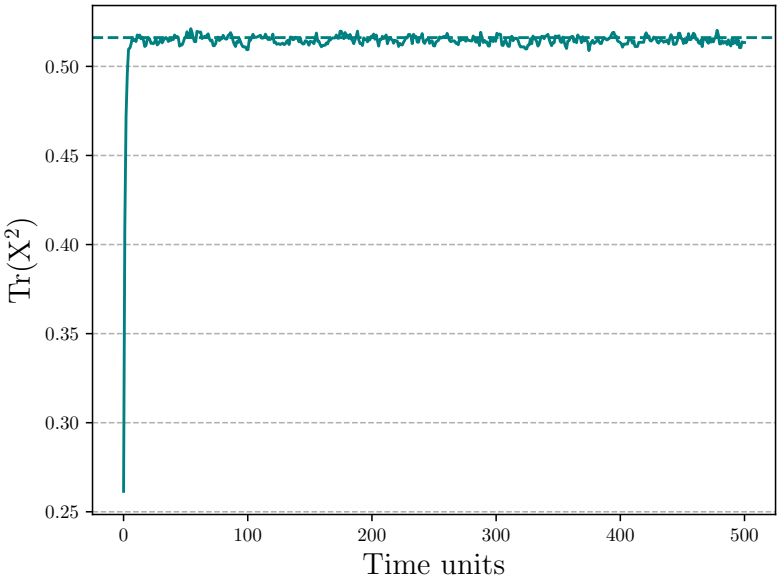

Figure 6. The computed value of $\mathrm{Tr}(X^2)$ with MC methods is consistent with that obtained using analytical saddle-point methods (shown by dashed lines). These results are with $N = 300$ and $g = 1$.

It is only well-defined if $g \leq 0.21935$. This is the radius of convergence of the planar perturbation series. Though this method can be exactly solved, we encourage the reader to attempt Exercise **...** to numerically solve this using MC. This exercise is good practice on how we can modify the potential in the given codes to study another model of interest.

• Exercise 7: Check that the one matrix model given in the appendix also reproduces the correct result for cubic potential i.e., $V(M) = M^2/2 + gM^3$. The result from bootstrap can be found in Ref. [17].

There is another interesting dimer matrix model which was used to understand the Yang-Lee edge singularity [19] in Ising model on random surface. It is given by:

$$Z = \int \mathcal{D}X\mathcal{D}M \exp N\mathrm{Tr}\left(-\frac{X^2}{2} + \frac{gX^4}{4} - \frac{M^2}{2} + g\sqrt{\zeta}MX^3 \right) \tag{3.16}$$

and then if we integrate out the matrix M to get:

$$Z = \int \mathcal{D}X \exp N\mathrm{Tr}\left(-\frac{X^2}{2} + \frac{gX^4}{4} + g^2\zeta\frac{X^6}{2} \right) \tag{3.17}$$

Note that if we set $\zeta = 0$, this reduces to the familiar 1MM problem we solved in previous section.

• Exercise 8: Check that (3.17) follows from (3.16) and modify the potential for the 1MM PYTHON code to study this model.

3.4 Hoppe-type matrix models

This model was first introduced by Hoppe in 1982 and solved later by Kazokov-Kostov-Nekrasov (KKN) in [20]. The partition function is given by:

$$Z = \int \mathcal{D}X\mathcal{D}Y \exp\Big[- N \ \mathrm{Tr}(\mathrm{X}^2 + \mathrm{Y}^2 - \mathrm{g}^2[\mathrm{X},\mathrm{Y}]^2)\Big] \tag{3.18}$$

At large values of coupling i.e., $g \to \infty$, this model becomes commuting with $[X, Y] \to 0$. The presence of commutator term in matrix models is common especially in models which have a dual gravity interpretation. The exact result for average action is:

$$2\langle S_c\rangle + \langle S_q\rangle = N^2 - 1, \tag{3.19}$$

where $S_c = -Ng^2\mathrm{Tr}[\mathrm{X},\mathrm{Y}]^2$ and $S_q = N\mathrm{Tr}(\mathrm{X}^2 + \mathrm{Y}^2)$ This average action serves as a good check of the code.

We can alternatively also consider a slightly more general two-matrix model which reduces to Hoppe's model mentioned above in a special limit. Such a matrix model is generally not solvable. This model was considered in [17] and solved using bootstrap methods.

$$Z = \int \mathcal{D}X\mathcal{D}Y \exp\Big[- N \ \mathrm{Tr}(X^2 + Y^2 - h^2[X,Y]^2 + gX^4 + gY^4)\Big] \tag{3.20}$$

For $h = 0$ it can be reduced to Hoppe's matrix model which can be solved via saddle-point analysis or through the reduction to a Kadomtsev-Petviashvili (KP) type equation. For $g = 0$ it reduces to two decoupled one-matrix models and for $g = \infty$, we have [X, Y] = 0 and it becomes an eigenvalue problem.

We show in Fig. 7 that the results obtained using Monte Carlo (MC) are consistent with those obtained in [17]. The MC results were obtained for $N = 300$ in about 2000 seconds on a 2019 Macbook Model A1989. We also show the eigenvalue distribution for this model in . We give the PYTHON code in the Appendix and is also available at:

https://github.com/rgjha/MMMC

We ow consider a case where $g = 0$ ad $h = h_c = -0.049...$, the bootstrap results were to Very accurate around this point. We obtained MC results fort his car

fill here

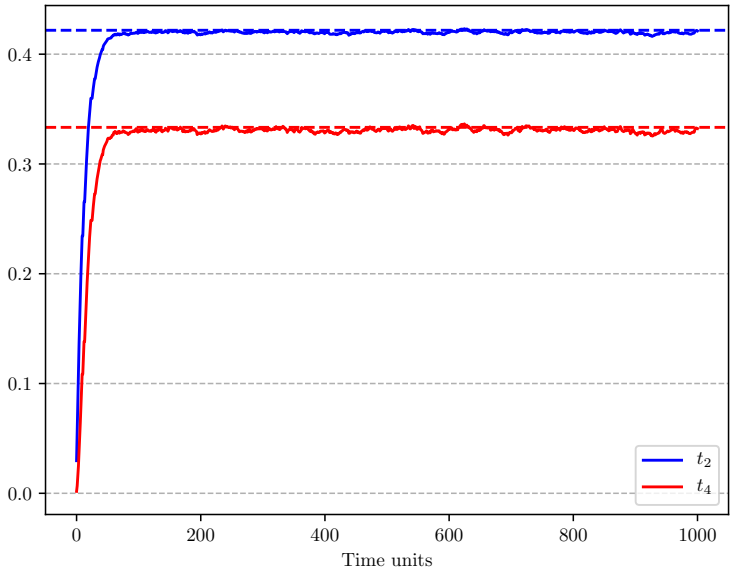

Figure 7. The matrix model defined by 3.20 is not solvable and was recently studied using bootstrap methods. We show the MC by solid lines and those obtained using bootstrap by dashed lines. We get about three digits of accuracy with 1000 time units by running for 1.7 hours on the laptop. The results are with $g = h = 1$ and $N = 300$. For larger N say, $N = 800$, the same run will take about 17 hours.

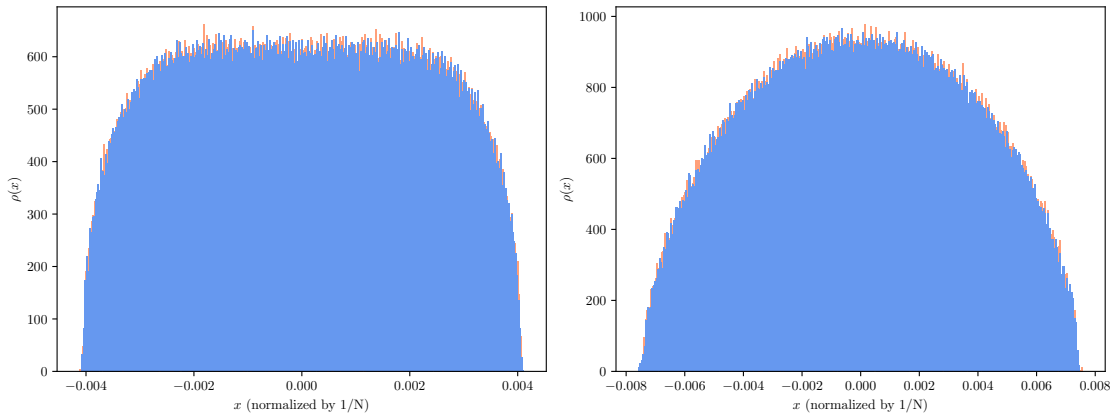

Figure 8. Left: The eigenvalue distribution of two matrices for $g = h = 1$ with $N = 300$. Right: The distribution for $g = 0, h = -0.049....$ with $N = 300$.

It is interesting to consider the same matrix model by flipping the sigs of the quadratic terms in X, Y i.e.,:

$$Z = \int \mathcal{D}X\mathcal{D}Y \exp\Big[- N \ \mathrm{Tr}(-X^2 - Y^2 - h^2[X,Y]^2 + gX^4 + gY^4)\Big] \tag{3.21}$$

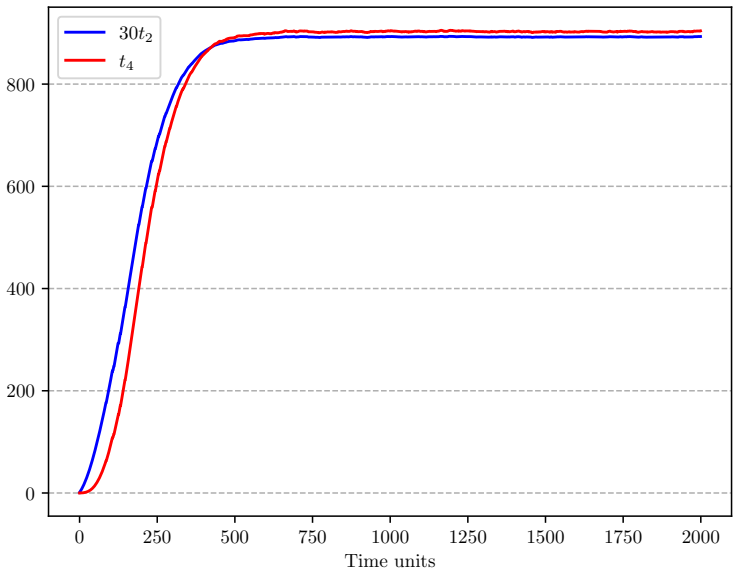

Figure 9. We show t_2 and t_4 obtained from MC. We have rescaled t_2 to table viewing on same plot. The values we obtain for $N = 300$ is $t_2 = 29.73(3)$ and $t_4 = 903(3)$. This took about 3.5 hours on a laptop since it thermalizes late and we need to run bit longer.

This results in the $\mathbb{Z}_2$ symmetry being broken. This model was considered using bootstrap methods in [] but compared to the symmetric case, it was tough to get accurate results because of slow convergence of the method. To benchmark our MC results, we explored this model with the set of couplings considered in the work above and obtain precise values. The results are shown on the figure taken from permission from [].

Note that while the results for t_2 and t_4 are pretty robust, it is less so for t_1 and seems to converge to different answers as shown in the figure.

This is shown in Fig. 11[8]

We also show the eigenvalue distribution of the matrices X, Y.

3.5 Extension of Hoppe's model to D matrices

After our discussion on models involving one or two matrices, we now turn to matrix models with equal or more than three matrices. Matrix model with $SO(D)$ invariance:

[8] We thank Zechuang Zhang and Vladimir Kazakov for discussion about this and for allowing us to use the figure from their paper.

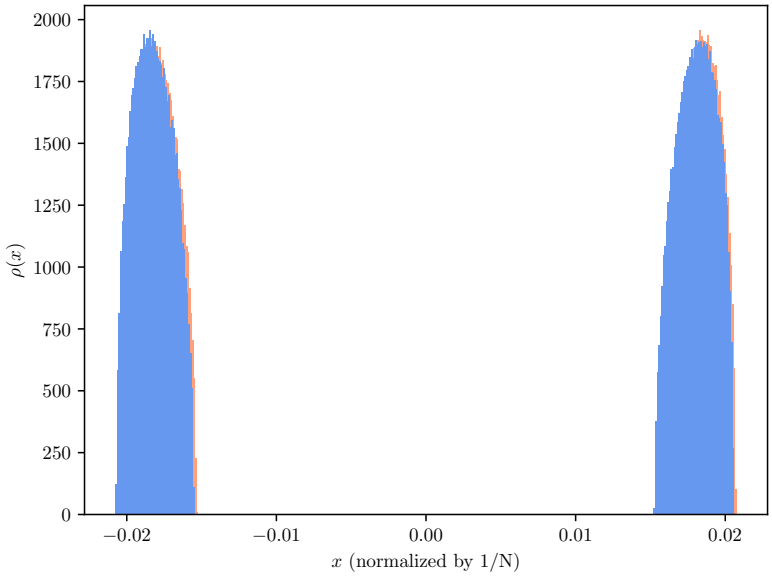

Figure 10. The eigenvalue distribution of two matrices for $g = 1/30$ and $h = 1/15$ with $N = 300$ for the potential given in 3.21.

$$Z = \int \mathcal{D}X_1 \cdots \mathcal{D}X_d \ \exp\Big[- Nh \sum_i \mathrm{Tr} X_i^2 + \frac{N\lambda}{4} \sum_{i \neq j} \mathrm{Tr}[X_i, X_j]^2 \Big] \tag{3.22}$$

If we consider $X_i \mapsto (1+\epsilon)X_i$, it must leave Z invariant, one arrives at the following exact relation:

$$D(N^2-1) = 2h\langle \mathrm{Tr} X_i^2 \rangle - N\lambda \langle \mathrm{Tr}[X_i, X_j]^2 \rangle. \tag{3.23}$$

This serves as a good check of the code and is satisfied to an accuracy of 0.02-0.03% after ignoring thermalization cut. We studied this model using MC for $D = 3, 5$ with $h = 1, \lambda = 1$ and compute:

$$\langle R^2 \rangle = \frac{1}{DN} \Big\langle \mathrm{Tr} \sum_{i=1}^{D} X_i^2 \Big\rangle, \quad \langle R^4 \rangle = \frac{1}{DN} \Big\langle \mathrm{Tr} \sum_{i=1}^{D} X_i^4 \Big\rangle. \tag{3.24}$$

The results are collected in Table 2.

• Exercise 9: Study the model defined by (3.22) for $d = 3$ by modifying the code given in the appendix for studying Yang-Mills type model defined by (3.26). Check that you obtain results consistent with that in Table 2.

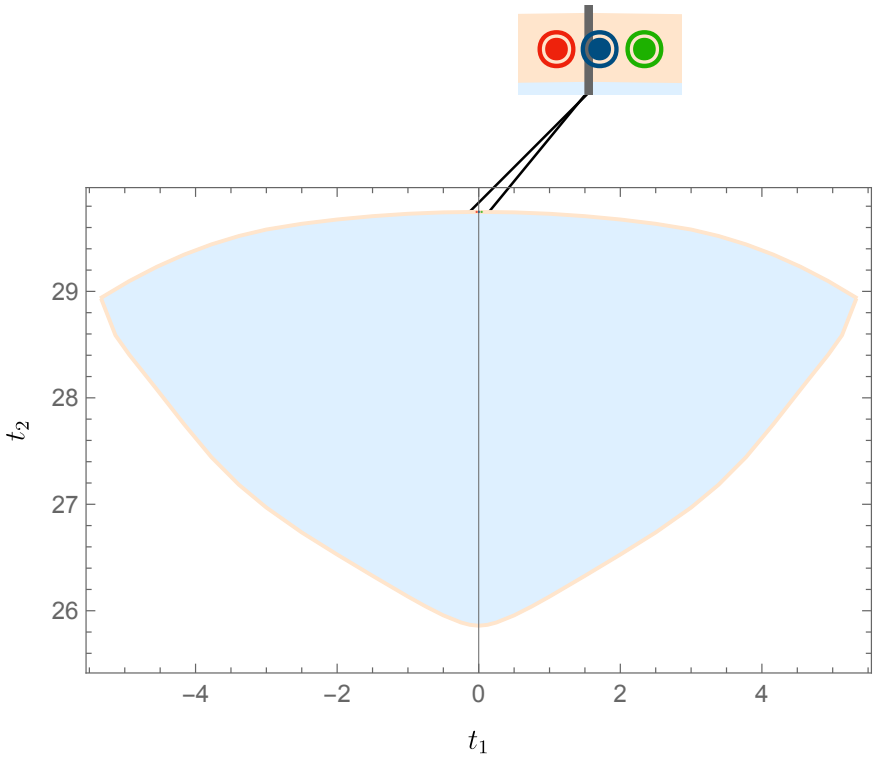

Figure 11. We show that the MC results from different streams (fresh starts) give different results for t_1 but t_2 and t_4 is unchanged within errors. This is not fully consistent with the region obtained in [17]. We checked the results for $N = 50$ and $N = 500$ and still never get $t_1 > 0.10$.

D	$\langle R^2 \rangle$	$\langle R^4 \rangle$
3	0.279(4)	0.158(5)
5	0.212(3)	0.091(5)

Table 2. The results obtained for $D = 3, 5$ YM matrix models with mass terms are given for $\lambda = 4, h = 1$ with $N = 300$.

3.6 Closed matrix chain models

The matrix chain is a complicated p matrix model which was first considered in [13]. When the chain is connected, the model is *not solvable* with $p \geq 4$.

3.7 $p = 3$ closed chain

We use Monte Carlo methods to study this with $p = 3, 4$ with $N = 300$ which is close to the desired planar limit. It would be good if this four matrix model can also be *bootstrapped* in coming years. The MC code to solve this model is available at:

https://github.com/rgjha/MMMC

$$Z_p(g,c,\kappa) = \int \mathcal{D}M_1 \cdots \mathcal{D}M_p \exp \mathrm{Tr}\left(\sum_{i=1}^{p} -M_i^2 - gM_i^4 + c\sum_{i=1}^{p-1} M_i M_{i+1} + \kappa M_p M_1 \right) \quad (3.25)$$

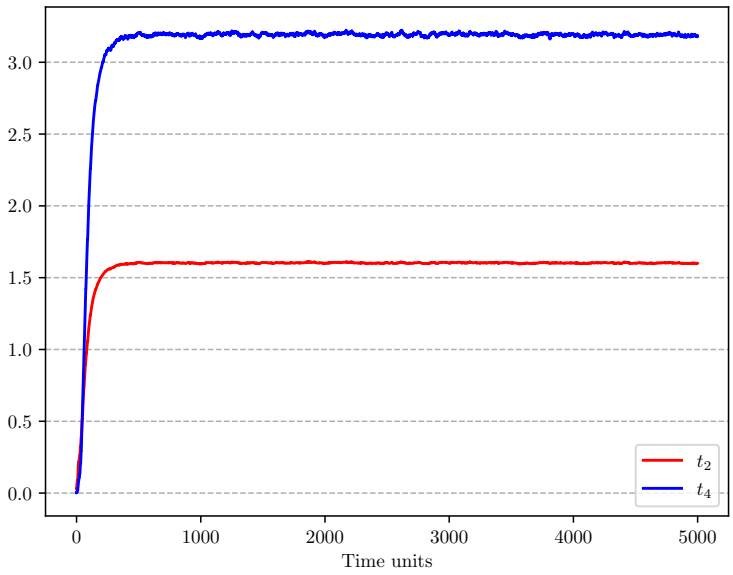

Figure 12. We find that for three matrix closed model we find $t_2 = 1.603(5)$ and $t_4 = 3.193(5)$. This is for $g = 1, c = \kappa = 1.35$ for $N = 300$. We have computed errors after discarding first 1000 time units and using jackknife blocking. Note that for this set of parameters it seems like $t_2 = t_4/2$. We found that for $g = 2, c = \kappa = 1.35$, $t_2 = 0.775(2)$ and $t_4 = 0.887(3)$. It is straightforward to understand the behaviour as a function of g at fixed c, κ.

It is easy to explore the sign of $\mathcal{O}(1/N)$ corrections using Monte Carlo. The simplest way is to do another set of simulation at $N = 50$ and see how t_2 and t_4 change. For this model, the numerics suggest that the sign is negative i.e., $t_2 \sim 1.603(1 - \frac{A}{N^2})$

3.8 $p = 4$ closed chain

We can study a slightly complicated model with four matrices with periodic (closed) condition as defined in 3.25. This code can be obtained by simply changing `NMAT=3` to `NMAT=4`. The results are collected in Table 3. Note that setting $\kappa = 0$ will reduce to open chain.

Generalize code

g	t_2	t_4
1	0.741(2)	0.825(3)
2	1.577(2)	3.093(3)

Table 3. The obtained results for $g = 1, 2$ four matrix closed chain with $N = 300, c = \kappa = 1.35$.

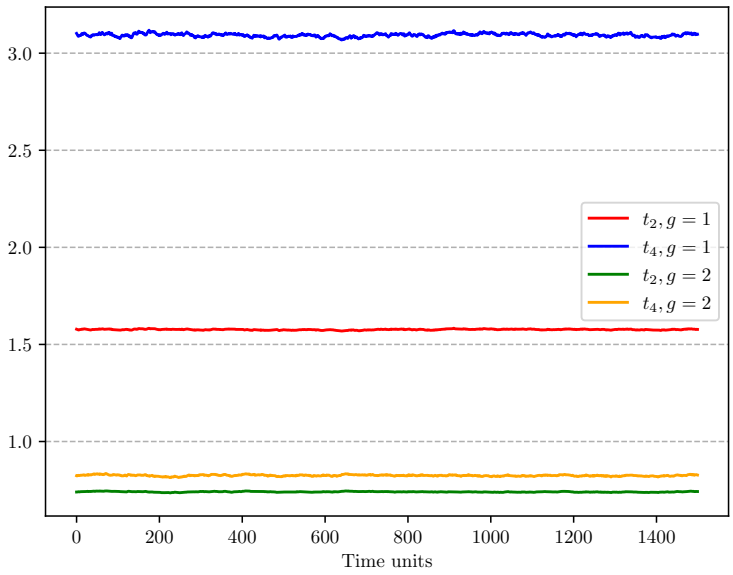

Figure 13. We show the expectation value for $N = 300, c = \kappa = 1.35$ with two different g. As the attentive reader will note, this is not a `fresh` start. See Appendix D for details.

3.9 Three and more matrix models: Yang-Mills matrix models

In Sec. 3.5 we discussed geeralization of Hoppe type matrix models to D matrices. It is also interesting to consider these models when $h = 0$ with D matrices. We refer to these models as 'Yang-Mills' type models following Refs.[21, 22]. We refer the reader to these references for more details.

$$S = \frac{N}{4\lambda} \int \mathrm{Tr}\left(\sum_{\mu \neq \nu} [X_\mu, X_\nu]^2 \right) \tag{3.26}$$

The MC code to solve this model is available at:

`https://github.com/rgjha/MMMC`

This is just a general version of the well-known bosonic part of IKKT model where $D = 10$. But, this model can be studied for any D and has interesting features, see Ref. [23].

Shortly after the BFSS matrix model[9] was proposed as a description of M-theory, the authors of [28] proposed a reduction of quantum mechanical model down to zero dimensions which is now named after them. This model is related to D-instantons in the sense that it has the same action as that of low-energy effective action of D-instantons. This model was proposed with hope to play a key role in description of Type IIB string theory in DLCQ. Though a complete large N solution is out of reach, there is lot of numerical results available. We also note that there has been some recent work which tries to take the master-field approach to the IKKT model [29]. This is a promising direction but it is not yet clear how effective it is in general. The action for this model is schematically written as:

$$S = \frac{N}{4\lambda} \int \mathrm{Tr}\Big(\frac{1}{4}[X_\mu, X_\nu]^2 + \overline{\psi}\Gamma^\mu[A_\mu, \psi]\Big) \tag{3.27}$$

where X_μ and ψ are $N \times N$ Hermitian matrices. The gauge field is a ten-dimensional vector and the ψ are ten-dimensional Majorana-Weyl spinor field respectively. This model in zero dimensions possess no usual space-time supersymmetry but is dimensional reduction of $\mathcal{N} = 1$ SYM theory in ten dimensions. It is expected that in this model both space and time should be dynamically generated as a result of the dynamics of the large matrices. The space-time should be constructed only from the matrices. IKKT model has no free parameters since λ can be absorbed in the field redefinitions.

In the original model, since it is a ten-dimensional reduction we have, $\mu, \nu = 1 \cdots 10$. It is opossible to consider other models where $\mu, \nu = 1 \cdots D$ where $D = 4, 6$. One might worry whether because of the integral measure being over non-compact X, it is divergent. The convergence issues of these models for $D = 4, 6, 10$ was studied by ——. In what follows in these notes, we will ignore the fermionic term and only focus on the commutator/bosonic term. One of the observables (also known as 'size' or i.e., extent of scalars) which we compute in these models is:

It is known that $\langle R^2 \rangle$ should behaves as $\sqrt{\lambda}$ in the large N limit from **Nishimura '98** but we have not found any previous study which fixes the coefficient. Our numerical results suggests that $\langle R^2 \rangle = 0.361(2)\sqrt{\lambda}$ and $\langle R^4 \rangle = 0.266(3)\lambda$ with $N = 300$ for a wide range of couplings i.e., $\lambda \in [1, 100]$. We also note that we find the $\sqrt{\lambda}$ and λ behaviour for $\langle R^2 \rangle$ and $\langle R^4 \rangle$ valid down to $D = 3$. One of the standard tests we do for the reliability of the numerical results is computing the average action. It can be shown that under a change: $X \to e^\epsilon X$ if

[9] This matrix model was proposed in Ref. [24] by Banks, Fischler, Shenker, and Susskind. This proposal related the uncompactified eleven dimensional M-theory in the light cone frame and the planar limit of the supersymmetric matrix quantum mechanics describing $D0$-branes. This model has been well-studied using numerical MC methods []. The publicly available code to study this model and its mass deformation (BMN matrix model) and higher-dimensional systems which describes D1/D2 branes [25–27] is available at https://github.com/daschaich/susy, while the highly efficient parallelized code over number of colours for only BFSS and BMN is available at https://sites.google.com/site/hanadamasanori/home/mmmm. The full discussion of these models is outside the purpose of this article.

we demand that Z is invariant, then we find:

$$\frac{\langle S \rangle}{N^2-1} = \frac{D}{4} \tag{3.28}$$

This is an exact result and is used to check the simulations. We show in Fig. 14 that for $D = 3, 5, 10$ we get the correct result and hence the PYTHON code above can be fully trusted. The timings for generating 500 time units/trajectories with $N = 300$ is about `50000, 2300, 2000` seconds for $D = 10, 5, 3$ respectively on 2.4 GHz i5 A1989 Mac. The results are collected in the table below.

D	$\langle R^2 \rangle$	$\langle R^4 \rangle$
3	1.129(3)	2.71(2)
5	0.608(2)	0.765(3)
10	0.361(2)	0.266(3)

Table 4. The results obtained for various D YM matrix models are given for $\lambda = 1$ with $N = 300$.

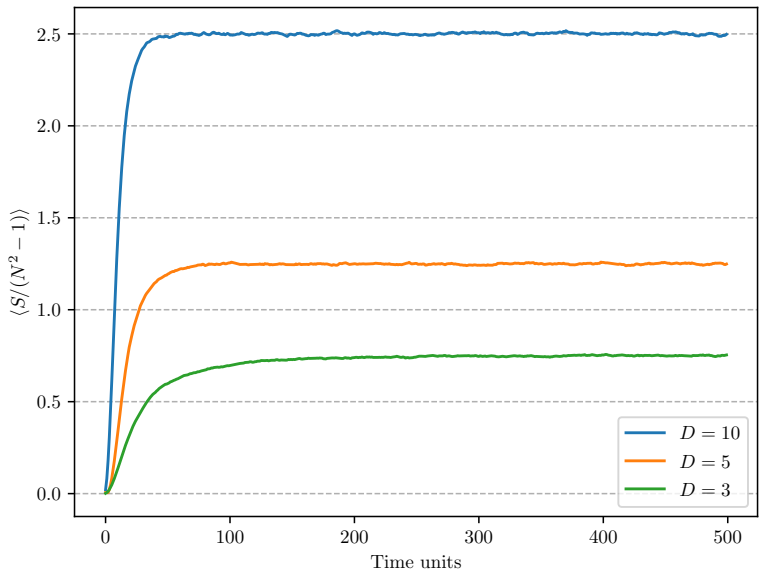

Figure 14. The average action (normalized) for the $D = 10$ bosonic sector of IKKT model.

• Exercise 10: Derive (3.28) by doing the change: $X_\mu \mapsto e^\epsilon X_\mu$ and ignoring terms in $\mathcal{O}(\epsilon^2)$

• Exercise 11: Carry out the MC computation for the bosonic Yang-Mills type matrix model with $D = 6$. After sufficient thermalization cut, check that the results are consistent with exact result obtained using Schwinger-Dyson equations within errors. Refer to appendix for the PYTHON code and to perform jackknife error analysis with sufficiently large block size. What is the ratio $\langle R^4 \rangle / \langle R^2 \rangle$ for fixed $\lambda = 1$.

SECTION 4

Summary

In these notes, we have discussed wide range of matrix models and described the Monte Carlo methods to solve them. We have obtained few new results and confirmed well-known solutions and recent bootstrap results. We hope this introduction will help interested readers to carry out these numerical computations on their laptop using the codes provided in the article and will eventually result in new ways to solve matrix models and to also bootstrap the models not yet explored.

Acknowledgements

The author is indebted to Pedro Vieira for discussions and for encouragement. We thank Vladimir Kazakov ad Zhechuag Zhang for email correspodece. The author is supported by a postdoctoral fellowship at the Perimeter Institute for Theoretical Physics. Research at Perimeter Institute is supported in part by the Government of Canada through the Department of Innovation, Science and Economic Development Canada and by the Province of Ontario through the Ministry of Economic Development, Job Creation and Trade. The author would like to thank Department of Science & Technology, Govern- ment of India for KVPY (Kishore Vaigyanik Protsahan Yojana) Scholarship during 2008-2010, European Union for Erasmus Mundus scholarship during 2010-2011, Department of Physics at Syracuse University for support during 2013-2019.

SECTION A

Orthogonal polynomials

One of the methods we discussed for the solution of large N limit of matrix models was the saddle point approximation. However, this method is not useful to understand the terms subleading in $1/N$. The method of orthogonal polynomials introduced by Bessis [**?**] is usually used for such computations. In fact, it is also very useful in solving multi-matrix models. These polynomials are defined as:

$$\int d\lambda e^{-V(\lambda)} P_n(\lambda) P_m(\lambda) = \int d\mu(\lambda) P_n(\lambda) P_m(\lambda) = a_n \delta_{mn} \tag{A.1}$$

where $d\mu(\lambda) = d\lambda e^{-V(\lambda)}$ is the measure

The basic idea is to rewrite the Vandermonde determinant appearing after we change from matrix basis to the basis of eigenvalues.

$$\Delta(\lambda) = \det(\lambda_i^{j-1})_{1\leq i,j\leq N} = \det(P_{j-1}(\lambda_i))_{1\leq i,j\leq N} \tag{A.2}$$

These polynomials can solve:

$$\exp(Z) = \int dM \exp[-\mathrm{Tr}V(M)] \tag{A.3}$$

In fact, it can be shown that A.3 is equivalent to:

$$\exp(Z) = N!\ a_0^N \prod_{k=1}^{N} f_k^{N-k} \tag{A.4}$$

where $f_k := a_k/a_{k-1}$. Hence, solving the matrix model is now equivalent to solving for the normalization appearing in A.1.

As we have shown in ...., we can use this method to solve the unitary matrix model.

Ising Model on random graph was studied as two-matrix model first by Kazakov in 1986 and the partition function is given by:

$$Z = \int \mathcal{D}A\mathcal{D}B \exp N\mathrm{Tr}\bigg(- A^2 - B^2 + 2cAB - g\frac{A^3}{3} - g\frac{B^3}{3}\bigg) \tag{A.5}$$

Note that this has $\mathbb{Z}_2$ symmetry because of Z being invaiant under $A \mapsto B$

$$Z = \int \mathcal{D}A\mathcal{D}B \exp N\mathrm{Tr}\bigg(- A^2 - B^2 + 2cAB - g_A e^h\frac{A^3}{3} - g_B e^{-h}\frac{B^3}{3}\bigg) \tag{A.6}$$

In 1986, Kazakov solved the Ising model on a random graph by **nice word** it through a two-matrix model [**?**]. This was later extended by Boulatov and Kazakov to admit magnetic fields [**?**]. We will not discuss the entire solution but will sketch the solution. They also computed the critical exponents and found different results than Onsager's case for regular square lattice. The exponents computed satisfied the usual Rushbrooke's law ($\alpha+2\beta+\gamma = 2$) and Widom's scaling law: $\gamma/\beta = \delta - 1$. These values coincide with the exponents obtained in a three-dimensional spherical model. This is a striking correspondence between exponents of two different models in different dimensions! In fact, after few years, while discussing the Yang-Lee edge sigularity (YLES) on dyamical graph, it was shown that an additional exponent $\sigma = 1/2$ also behaved accordingly. We have listed the exponents in Table (5). ggg

Crit. exponents	Ising model on random planar graph	Ising model on regular lattice
α	-1	0
β	1/2	1/8
γ	2	7/4
δ	5	15
νd	3	2
$\gamma_{\rm str}$	-1/3	-

Table 5. Summary of critical exponents obtained for two-dimensional Ising model on different graphs

By turning the Z given in .... in terms of eigenvalues, we get:

$$Z = \int dX dY \Delta(X)\Delta(Y) \exp\Big[- N \sum_i (x_i^2 + y_i^2 + 2c x_i y_i + 4g e^h x_i^4 + 4g e^{-h} y_i^4)\Big] \tag{A.7}$$

It is now clear that we would need two polynomials $P_k(x)$ and $Q_j(y)$ for this case such that their determinant matches $\Delta(X)$ and $\Delta(Y)$ respectively. These polynomials satisfy the following orthonormal condition:

$$\int dx dy e^{-NV(x,y)} P_k(x) Q_j(y) = h_k \delta_{kj} \tag{A.8}$$

They also satisfy several recursion relations for which the interested reader can refer to [**?**]:

$$Z = \int dX dY \det[P_r(x_k)] \det[Q_r(y_k)] \exp\Big[- N \sum V(X,Y)\Big] \tag{A.9}$$

where we have denoted $\sum_i (x_i^2 + y_i^2 + 2c x_i y_i + 4g e^h x_i^4 + 4g e^{-h} y_i^4)$ by $V(X,Y)$. Transforiming to the eigenvalue basis of both matrices X and Y and using the expansion of the determinant we get:

$$\begin{aligned} Z &= \epsilon^{i_1 \cdots i_N} \epsilon^{j_1 \cdots j_N} \int dx_{1\cdots N} dy_{1\cdots N} P_{i_1}(x_1) \cdots P_{i_N}(x_N) Q_{j_1}(y_1) \cdots Q_{j_N}(y_N) e^{-N \sum V(x_i, y_i)} \\ &= \epsilon^{i_1 \cdots i_N} \epsilon^{j_1 \cdots j_N} \prod_{r=1}^{N} \int dx_r dy_r e^{-NV(x_r,y_r)} P_{i_r}(x_r) Q_{j_r}(y_r) \\ &= N! \prod_{i=0}^{N-1} h_i \end{aligned} \tag{A.10}$$

We can define $f_k := h_k / h_{k-1}$ and hence (A.10) implies:

$$\log Z_N(c,g,h) = \log N! + N\log h_0 + \sum_{k=1}^{N-1}(N-k)\log f_k \tag{A.11}$$

One is usually interested in computing the quantity (the subscript 'pc' denotes planar/continuum limit i.e., $N \to \infty$):

$$F_{pc} = \frac{1}{N^2}\log\left(\frac{Z(c,g,h)}{Z(c,0,0)}\right) = \frac{1}{N}\sum_{k=1}^{N-1}\left(1 - \frac{k}{N}\log\left(\frac{f_k}{f_{k,0}}\right)\right) \tag{A.12}$$

SECTION B

Mathematica code for solution of one-matrix model

We now give the details to solve the one matrix model in MATHEMATICA. For this we consider the potential:

$$V(Y) = \frac{Y^2}{2} + \frac{gY^4}{4}$$

As we have shown in the text, for this case, the higher moments of trace of Y are related and hence we will just calculate $\mathrm{Tr}Y^2$ in the planar limit (normalized by N). We give the code below for $g = 1$. The reader is encouraged to try and change g and see how the results change.

```
V[y_]=y^2/2+(g y^4)/4;
G[x_]=Integrate[-1/(2\[Pi]I)Sqrt[x^2-a^2]/Sqrt[y^2-a^2](N V'[y])/(x-y),{y,-a,a},
Assumptions->{x>a,a>0}];
sol=Series[G[x],{x, \[Infinity], 1}]-N/x//Simplify//Solve[# == 0,a]&//Simplify;
Series[G[x],{x,\[Infinity], 5}]//Normal;
{Coefficient[%, x, -3]}/N;
% /. sol;
%/.{g -> 1}//Chop//N// Grid
```

SECTION C

Brief explanation and comments on running the Python code

We provide codes which can deal with several different types of Hermitian matrix models with real action. The codes we give in this appendix and some which are not here are all located at:

https://github.com/rgjha/MMMC

These codes can also be modified to other potentials as required. In general, only two routines need modifications. One is `def potential(X)` and other is the `def force(X)`. The first

involves (mostly) trace of product of matrices and the second is the derivative of those matrix traces. These codes in PYTHON is rather terse and are all under `300` lines each while about $\sim 85\%$ of them are common things like: leapfrog integrator, Metropolis step, saving/reading configuration file, and plotting the data. We have tried no optimization to make it efficient and the only motivation is that someone who has never ran a MC code can do so quickly and use it as guidance for deriving exact results or for bootstrapping purposes. We need to take care of few things which are mentioned below.

- The acceptance rate should always be more than 50% on average. If the acceptance rate is less than this, we must reduce the size of leapfrog integrator by reducing `dt` in the global definitions at the starting. Note that reducing this time step makes the computation little more expensive. This time step needs to be modified accordingly if you want to explore values of N much more than $N = 300$ which we mostly use in these notes. The code will give a warning if the acceptance falls below 50%. The optimum acceptance for the integrator we use is about 60%.

- We must not change the step size during the entire time of simulation. It has to be chosen to a value where acceptance is reasonable and then kept constant. If the acceptance doesn't improve eve after reducing the time step, it signals an error in the force term.

- As a thumb rule, during the evolution, the `delta S` should fluctuate around zero with both negative and positive signs. However, this might not be true from the start, and should be monitored after sufficient thermalization. After thermalization, we should have $\langle e^{-\Delta H} \rangle = 1$ within errors. You can see why this is true in the section on solution to the selected exercises.

- We usually start a run by setting all matrices to zero (also referred sometimes as 'fresh'). Then as the evolution progresses, we store a new configuration by rewriting older one every 10 time units. The configuration file stores $N \times N$ matrices over which we do the matrix integral in binary format as `numpy array`. The size of this file can vary from few MB unto 50 MB or more depending on `NMAT` and `NC`. If you are not doing the run for the first time, it is better if you read in the configuration file as this will be more efficient. Note that this can only be done if `NMAT` and `NC` are the same or else it will throw some error. This can be modified easily to suit user's requirements.

- The code produces output files such as `t2*.txt` and `t4*.txt` which are moments of the matrices. The number of columns in these files will be equal to number of different matrices you considered in the potential i.e., `NMAT`. If you consider a ten matrix model (maximum we have used in these notes), these files will have ten corresponding columns.

- It is common practice among those who use Monte Carlo to never measure an observable every time unit (because of autocorrelation), but for these simple models, it is probably okay to do so. However, if you have enough computer time and want to do a systematic study, you should increase it. This is controlled by `GAP`. Another way to make sure that this is accepted for is to use a sufficiently large block size when computing errors using jackknife method.

- Note that Monte Carlo is a sampling method and hence we will always have errors for the expectation values. The errors must be carefully computed using either jackknife binning or some other method.

The list of code we make available is summarized below:[10]

1. One matrix model: https://github.com/rgjha/MMMC ⋆
2. Two matrix Hoppe-type models: https://github.com/rgjha/MMMC
3. Three and four matrix chain models (open and closed): https://github.com/rgjha/MMMC
4. Yang-Mills type models for different D: https://github.com/rgjha/MMMC ⋆
5. Yang-Mills type models with mass term for different D: This is Exercise () and the hint on how to proceed is given in Appendix E.

Though these codes have been checked several times over and compared to known solutions (where available), it is possible that there might still be minor bugs in them. If you encounter a problem or have questions, please contact the author.

SECTION D

Python code for solution of one-matrix model.

We now solve the 1MM using the Monte Carlo method which is the standard in lattice field theory literature. By executing the code given below on a modern laptop, we get the result shown in Fig. 6. We can readily extend this code (by changing `NMAT`) to study matrix models where the integration is over several different matrices. As we will discuss later, for the case of IKKT model, there are 'ten' matrices involved and hence this code will be adjusted accordingly to study that model for given λ and N. To run this code using Mac/Linux system (assuming you have PYTHON installed) just type in terminal: `python 1MM_MC.py 0 1 300 200`. The code takes four arguments. The first two are binary arguments related to whether we are reading any old configuration file and whether we want to save the one which will be generated. `0 1` means that we are not reading any configuration but we want to save it for

[10] (⋆ denotes that is in given also in the appendix)

later use. The third argument is the size of the matrices to consider, in an ideal world, planar limit is $N \to \infty$ but here we have `N = 300`. The last argument is for how many trajectories of Markov chain Monte Carlo we want to run the simulation. To converge to the correct answer `500` should be enough. This should take about `1300` seconds on a standard modern laptop.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import time
import datetime
import sys
import numpy as np
import random
import math
from numpy import linalg as LA
from matplotlib.pyplot import *
from matplotlib import pyplot as plt

startTime = time.time()
print ("STARTED:" , datetime.datetime.now().strftime("%d %B %Y, %H:%M:%S"))

if len(sys.argv) < 6:
  print("Usage: python", str(sys.argv[0]), "READIN " "SAVE " "NCOL " "NITERS " "g ")
  sys.exit(1)

READIN = int(sys.argv[1])
SAVE = int(sys.argv[2])
NCOL = int(sys.argv[3])
Niters_sim = int(sys.argv[4])
g = float(sys.argv[5])

NMAT = 1
dt = 1e-3
nsteps = int(0.5/dt)
GAP = 1.

if Niters_sim%GAP != 0:
  print("'Niters_sim' mod 'GAP' is not zero ")
  sys.exit(1)

cut=int(0.25*Niters_sim)
X = np.zeros((NMAT, NCOL, NCOL), dtype=complex)
mom_X = np.zeros((NMAT, NCOL, NCOL), dtype=complex)
f_X = np.zeros((NMAT, NCOL, NCOL), dtype=complex)
X_bak = np.zeros((NMAT, NCOL, NCOL), dtype=complex)
HAM, expDS, trX2, trX4, MOM = [], [], [], [], []

print ("Matrix integral simulation with %2.0f matrix"%(NMAT))
print ("NCOL = " "%3.0f " "," " and g = " " %4.2f" % (NCOL, g))
print
    ("--------------------------------------------------------------------------------")
```

```
def dagger(a):
   return np.transpose(a).conj()

def box_muller():
   PI = 2.0*math.asin(1.0);
   r = random.uniform(0,1)
   s = random.uniform(0,1)
   p = np.sqrt(-2.0*np.log(r)) * math.sin(2.0*PI*s)
   q = np.sqrt(-2.0*np.log(r)) * math.cos(2.0*PI*s)
   return p,q

def copy_fields(b):
   for j in range(NMAT):
      X_bak[j] = b[j]
   return X_bak

def rejected_go_back_old_fields(a):
   for j in range(NMAT):
      X[j] = a[j]
   return X

def refresh_mom():
   for j in range (NMAT):
      mom_X[j] = random_hermitian()
   return mom_X

def random_hermitian():
   tmp = np.zeros((NCOL, NCOL), dtype=complex)

   for i in range (NCOL):
      for j in range (i+1, NCOL):
         r1, r2 = box_muller()
         tmp[i][j] = complex(r1, r2)/math.sqrt(2)
         tmp[j][i] = complex(r1, -r2)/math.sqrt(2)

   for i in range (NCOL):
      r1, r2 = box_muller()
      tmp[i][i] = complex(r1, 0.0)

   return tmp


def makeH(tmp):

   tmp2 = 0.50*(tmp+dagger(tmp)) - (0.50*np.trace(tmp+dagger(tmp))*np.eye(NCOL))/NCOL

   for i in range (NCOL):
      tmp2[i][i] = complex(tmp[i][i].real,0.0)

   if np.allclose(tmp2, dagger(tmp2)) == False:
      print ("WARNING: Couldn't make hermitian")
```

```
    return tmp2

def hamil(X,mom_X):
    ham = potential(X)
    for j in range (NMAT):
        ham += 0.50 * np.trace(np.dot(mom_X[j],mom_X[j])).real
    return ham

def potential(X):
    pot = 0.0
    for i in range (NMAT):
        pot += 0.50 * np.trace(np.dot(X[i],X[i])).real
        pot += (g/4.0)* np.trace(X[i] @ X[i] @ X[i] @ X[i]).real
    return pot*NCOL

def force(X):

    for i in range (NMAT):

        f_X[i] = (X[i] + (g*(X[i] @ X[i] @ X[i])))*NCOL

    for j in range(NMAT):
        if np.allclose(f_X[j], dagger(f_X[j])) == False:
            f_X[j] = makeH(f_X[j])

    return f_X

def leapfrog(X,dt):

    mom_X = refresh_mom()
    ham_init = hamil(X,mom_X)

    for j in range(NMAT):
        X[j] += mom_X[j] * dt * 0.5

    for i in range(1, nsteps+1):
        f_X = force(X)
        for j in range(NMAT):
            mom_X[j] -= f_X[j] * dt
            X[j] += mom_X[j] * dt

    f_X = force(X)
    for j in range(NMAT):

        mom_X[j] -= f_X[j] * dt
        X[j] += mom_X[j] * dt * 0.5

    ham_final = hamil(X,mom_X)
    return X, ham_init, ham_final

def update(X, acc_count):

    X_bak = copy_fields(X)
```

```
    X, start, end = leapfrog(X, dt)
    change = end - start
    expDS.append(np.exp(-1.0*change))
    if np.exp(-change) < random.uniform(0,1):
       X = rejected_go_back_old_fields(X_bak)
       print(("REJECT: deltaH = " "%8.7f " " startH = " "%8.7f" " endH = " "%8.7f" %
           (change, start, end)))
    else:
       print(("ACCEPT: deltaH = " "%8.7f " "startH = " "%8.7f" " endH = " "%8.7f" %
           (change, start, end)))
       acc_count += 1

    if MDTU%GAP == 0:
       tmp0 = np.trace(np.dot(X[0],X[0])).real
       trX2.append(tmp0/NCOL)
       tmp1 = np.trace((X[i] @ X[i] @ X[i] @ X[i])).real
       trX4.append(tmp1/NCOL)
       f3.write("%4.8f \n" %(tmp0/NCOL))
       f4.write("%4.8f \n" %(tmp1/NCOL))

    return X, acc_count

if __name__ == '__main__':


    if READIN ==0:
       for i in range (NMAT):
          for j in range (NCOL):
             for k in range (NCOL):
                X[i][j][k] = complex(0.0,0.0)

    if READIN ==1:

       name_f = "config_1MM_N{}.npy".format(NCOL)
       A = np.load(name_f)

       for i in range (NMAT):
          for j in range (NCOL):
             for k in range (NCOL):
                X[i][j][k] = A[i][j][k]

       for j in range(NMAT):
          if np.allclose(X[j], dagger(X[j])) == False:
             print ("Input configuration 'X' not hermitian, ", LA.norm(X[j] -
                 dagger(X[j])), "making it so")
             X[j] = makeH(X[j])

       print ("Read old configuration file: ", name_f)


    f3 = open('t2_1MM_N%s_g%s.txt' %(NCOL,round(g,4)), 'w')
    f4 = open('t4_1MM_N%s_g%s.txt' %(NCOL,round(g,4)), 'w')
```

```
    acc_count = 0.

    for MDTU in range (1, Niters_sim+1):
       X, acc_count = update(X, acc_count)

       if (MDTU)%10 == 0 and SAVE == 1:

          name_f = "config_1MM_N{}.npy".format(NCOL)
          print ("Saving configuration file: ", name_f)
          np.save(name_f, X)

    f3.close()
    f4.close()

    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')
    MDTU = np.linspace(0, int(Niters_sim/GAP), int(Niters_sim/GAP), endpoint=True)
    plt.ylabel(r'Tr(X$^2)/N$',fontsize=12)
    plt.xlabel('Time units', fontsize=12)
    plt.grid(which='major', axis='y', linestyle='--')
    plt.axhline(y=0.516151, color='teal', linestyle='--')
    plt.figure(1)
    plot (MDTU, trX2, 'teal')
    print ("Fraction accepted", (acc_count/Niters_sim)*100)

    if acc_count/Niters_sim < 0.50:
       print("WARNING: Acceptance rate is below 50%")

    if READIN == 0:
       trX2 = trX2[cut:]
       trX4 = trX4[cut:]
       expDS = expDS[cut:]

    print("<Tr X^2 / NCOL>", np.mean(trX2), "+/-", (np.std(trX2)/np.sqrt(np.size(trX2)
        - 1.0)))
    print("<Tr X^4 / NCOL>", np.mean(trX4), "+/-", (np.std(trX4)/np.sqrt(np.size(trX4)
        - 1.0)))
    print("exp(-deltaH)", np.mean(expDS), "+/-", np.std(expDS)/np.sqrt(np.size(expDS) -
        1.0))
    outname = "1MM_N%s_g%s" %(NCOL, g)
    plt.savefig(outname+'.pdf')
    #plt.show()
    print ("COMPLETED:" , datetime.datetime.now().strftime("%d %B %Y, %H:%M:%S"))
    endTime = time.time()
    print ("Running time:", round(endTime - startTime, 2), "seconds")
```

SECTION E

YM matrix model for any D: Python code

We now provide the Monte Carlo code using which we can study the bosonic part of any D matrix YM matrix model and especially $D = 10$ which is the bosonic sector of IKKT model. More details about the model and its relation to the non-perturbative formulations of string

theory can be found in Ref. —. The reader will note that most of the code is similar to the one matrix model.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import random
import os
import math
from matplotlib.pyplot import *
from matplotlib import pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
import numpy as np
import random
import itertools
from scipy.linalg import expm
from sympy import LeviCivita
from numpy import linalg as LA
import time
import datetime
import sys
startTime = time.time()
print ("STARTED:" , datetime.datetime.now().strftime("%d %B %Y %H:%M:%S"))

if len(sys.argv) < 7:
  print("Usage: python", str(sys.argv[0]), "READIN " "SAVE_or_NOT " "NCOL " "NITERS "
      "D " "LAMBDA ")
  sys.exit(1)

READIN = int(sys.argv[1])
SAVE = int(sys.argv[2])
NCOL = int(sys.argv[3])
Niters_sim = int(sys.argv[4])
NSCALAR = int(sys.argv[5])
LAMBDA = float(sys.argv[6])

if NSCALAR < 2:
   print ("Number of scalars must be at least two")
   sys.exit(1)

COUPLING = float(NCOL/(4.0*LAMBDA))
GENS = NCOL**2 - 1
dt=5e-4
nsteps = int(1e-2/dt)
GAP=1
t2 = np.zeros((NSCALAR),dtype=float)
t4 = np.zeros((NSCALAR),dtype=float)
X = np.zeros((NSCALAR, NCOL, NCOL), dtype=complex)
mom_X = np.zeros((NSCALAR, NCOL, NCOL), dtype=complex)
f_X = np.zeros((NSCALAR, NCOL, NCOL), dtype=complex)
X_bak = np.zeros((NSCALAR, NCOL, NCOL), dtype=complex)
HAM, expDS, ACT, scalar = [],[],[],[]

print ("Yang-Mills type matrix model simulation with %2.0f matrices" % (NSCALAR))
```

```
print ("NCOL = " "%3.0f " "," " and coupling = " " %4.2f" % (NCOL, COUPLING))
print
    ("--------------------------------------------------------------------------------")

def dagger(a):
   return np.transpose(a).conj()

def box_muller():
   PI = 2.0*math.asin(1.0);
   r = random.uniform(0,1)
   s = random.uniform(0,1)
   p = np.sqrt(-2.0*np.log(r)) * math.sin(2.0*PI*s)
   q = np.sqrt(-2.0*np.log(r)) * math.cos(2.0*PI*s)
   return p,q

def comm(A,B):
   return np.dot(A,B) - np.dot(B,A)

def unit_matrix():
   matrix = np.zeros((NCOL, NCOL), dtype=complex)
   for i in range (NCOL):
      matrix[i][i] = complex(1.0,0.0)
   return matrix

def copy_fields(b):
   for j in range(NSCALAR):
      X_bak[j] = b[j]
   return X_bak

def rejected_go_back_old_fields(a):
   for j in range(NSCALAR):
      X[j] = a[j]
   return X

def refresh_mom():
   for j in range (NSCALAR):
      mom_X[j] = random_hermitian()
   return mom_X

def random_hermitian():
   tmp = np.zeros((NCOL, NCOL), dtype=complex)
   for i in range (NCOL):

      for j in range (i+1, NCOL):
         r1, r2 = box_muller()
         tmp[i][j] = complex(r1, r2)/math.sqrt(2)
         tmp[j][i] = complex(r1, -r2)/math.sqrt(2)

   for i in range (NCOL):
      r1, r2 = box_muller()
      tmp[i][i] = complex(r1, 0.0)
   return tmp
```

```
def makeH(tmp):

   tmp = 0.50*(tmp+dagger(tmp)) - (0.50*np.trace(tmp+dagger(tmp))*np.eye(NCOL))/NCOL

   if np.allclose(tmp, dagger(tmp)) == False:
      print ("WARNING: Couldn't make hermitian")

   return tmp

def hamil(mom_X):
   s = 0.0
   for j in range (NSCALAR):
      s += 0.50 * np.trace(np.dot(dagger(mom_X[j]),mom_X[j]))
   return s.real

def potential(X):
   s1 = 0.0
   for i in range (NSCALAR):
      for j in range (i+1, NSCALAR):
         co = np.dot(X[i],X[j]) - np.dot(X[j],X[i])
         tr = np.trace(np.dot(co,co))
         s1 -= COUPLING*tr.real

   return s1

def force(X):

   tmp_X = np.zeros((NSCALAR, NCOL, NCOL), dtype=complex)
   for i in range (NSCALAR):
      for j in range (NSCALAR):
         if i == j:
            continue
         else:
            temp = comm(X[i], X[j])
            tmp_X[i] -= comm(X[j], temp)
      f_X[i] = 2.0*COUPLING*dagger(tmp_X[i])

   for j in range(NSCALAR):
      if np.allclose(f_X[j], dagger(f_X[j])) == False:
         print ("...")
         f_X[j] = makeH(f_X[j])

   return f_X

def leapfrog(X,mom_X, dt):
   for j in range(NSCALAR):
      X[j] += mom_X[j] * dt/2.0
   f_X = force(X)

   for step in range(nsteps):
      for j in range(NSCALAR):
         mom_X[j] -= f_X[j] * dt
         X[j] += mom_X[j] * dt
```

```
      f_X = force(X)

   for j in range(NSCALAR):
      mom_X[j] -= f_X[j] * dt
      X[j] += mom_X[j] * dt/2.0

   return X, mom_X, f_X

def update(X):
   mom_X = refresh_mom()
   s1 = hamil(mom_X)
   s2 = potential(X)
   start_act = s1 + s2
   X_bak = copy_fields(X)
   X, mom_X, f_X = leapfrog(X,mom_X,dt)
   s1 = hamil(mom_X)
   s2 = potential(X)
   end_act = s1 + s2
   change = end_act - start_act
   HAM.append(abs(change))
   expDS.append(np.exp(-1.0*change))

   if np.exp(-change) < random.uniform(0,1):
      X = rejected_go_back_old_fields(X_bak)
      print(("REJECT: deltaS = " "%8.7f " " startS = " "%8.7f" " endS = " "%8.7f" %
          (change, start_act, end_act)))
   else:
      print(("ACCEPT: deltaS = " "%8.7f " "startS = " "%8.7f" " endS = " "%8.7f" %
          (change, start_act, end_act)))

   ACT.append(s2)

   tmp = 0.0
   for i in range (0,NSCALAR):
      val = np.trace(X[i] @ X[i]).real/NCOL
      val2 = np.trace(X[i] @ X[i] @ X[i] @ X[i]).real/NCOL
      t2[i] = val
      t4[i] = val2
      tmp += val

   tmp /= NSCALAR
   scalar.append(tmp)

   if MDTU%GAP == 0:

      f3.write("%4.8f \n" % (s2/GENS))
      for item in t2:
         f4.write("%4.8f " % item)
      for item in t4:
         f5.write("%4.8f " % item)
      f4.write("\n")
      f5.write("\n")
```

```
    return X


if __name__ == '__main__':

    if READIN ==0:
        for i in range (NSCALAR):
            X[i] = random_hermitian()/(NCOL**2)

    if READIN ==1:

        name_f = "config_YM_N{}_l_{}_D_{}.npy".format(NCOL, LAMBDA, NSCALAR)
        if os.path.isfile(name_f) == True:
            print ("Reading old configuration file:", name_f)

            A = np.load(name_f)

            for i in range (NSCALAR):
                for j in range (NCOL):
                    for k in range (NCOL):
                        X[i][j][k] = A[i][j][k]

            for j in range(NSCALAR):
                if np.allclose(X[j], dagger(X[j])) == False:
                    print ("Input configuration not hermitian, making it so")
                    X[j] = makeH(X[j])

        else:
            print ("Can't find config. file for this NCOL and LAM")
            print ("Starting from fresh")
            for i in range (NSCALAR):
                X[i] = random_hermitian()

            X = X/(NCOL**2)

    f3 = open('action_N%s_D%s.txt' %(NCOL,NSCALAR), 'w')
    f4 = open('t2_N%s_D%s.txt' %(NCOL,NSCALAR), 'w')
    f5 = open('t4_N%s_D%s.txt' %(NCOL,NSCALAR), 'w')


    for MDTU in range (1, Niters_sim+1):
        X = update(X)

        if MDTU%10 == 0 and SAVE ==1:
            print ("Saving config.")
            name_f = "config_YM_N{}_l_{}_D_{}.npy".format(NCOL, LAMBDA, NSCALAR)
            print ("Saving configuration file: ", name_f)
            np.save(name_f, X)


    f3.close()
    f4.close()
    f5.close()
```

```
ACT = [x/GENS for x in ACT]

print("<S> = ", np.mean(ACT), "+/-", (np.std(ACT)/np.sqrt(np.size(ACT) - 1.0)))
print("<exp(-deltaH)> = ", np.mean(expDS), "+/-",
    np.std(expDS)/np.sqrt(np.size(expDS) - 1.0))
print ("COMPLETED:" , datetime.datetime.now().strftime("%d %B %Y %H:%M:%S"))
endTime = time.time()

# Plot results!
t2plot = plt.figure(1)
plt.rc('text', usetex=True)
plt.rc('font', family='serif')
MDTU = np.linspace(0, int(Niters_sim/GAP), int(Niters_sim/GAP), endpoint=True)
plt.ylabel(r'$\langle R^2 \rangle$')
plt.xlabel('Time units')
plot(MDTU, scalar, 'teal')
plt.grid(which='major', axis='y', linestyle='--')
act_plot = plt.figure(2)
plt.ylabel(r'$\langle S/(N^2-1) \rangle$')
plt.xlabel('Time units')
plt.axhline(y=NSCALAR/4.0, color='blue', linestyle='--')
plot(MDTU, ACT, 'blue')
plt.grid(which='major', axis='y', linestyle='--')
outname = "YM_N%s_D%s" %(NCOL,NSCALAR)
pp = PdfPages(outname+'.pdf')
pp.savefig(t2plot, dpi = 300, transparent = True)
pp.savefig(act_plot, dpi = 300, transparent = True)
pp.close()
print ("Running time:", round(endTime - startTime, 2), "seconds")
```

SECTION F

Computing error using jackknife for the data files from Monte Carlo

We give a simple code for computing statistical errors in PYTHON . The interested reader can find more details in Ref. [30]. The code can be used as follows from a terminal: `python jk_error.py t2.txt 1000 25 0`. This just means that we ask the code to take out first `1000` time units data for thermalization cut and we set the size of block to be `25`. The last argument tells the code which column to consider for averaging with `0` meaning the first column. To ensure that we have used reasonable thermalization cut, one can check for different cuts and see if the results are same within errors. One can do the same for the block size.

```
#!/usr/bin/python3
import sys
import numpy as np
import itertools
```

```
from math import *
data = []; data_tot = 0. ; Data = [] ; data_jack = []

if len( sys.argv ) > 4:
   filename = sys.argv[1]
   therm_cut = int(sys.argv[2])
   blocksize = int(sys.argv[3])
   which_column = int(sys.argv[4])

if len( sys.argv ) <= 4:
   print("NEED 4 ARGUMENTS : FILE THERM-CUT BLOCKSIZE COLUMN_TO_PARSE")
   sys.exit()

file = open(filename, "r")
for line in itertools.islice(file, therm_cut, None):

   line = line.split()
   if which_column > int(np.shape(line)[0])-1:
      print ("Column to average does not exist")
      sys.exit(1)
   data_i = float(line[which_column])
   data.append(data_i)
   data_tot += data_i
   n = len(data)

n_b = int(n/blocksize)
B = 0.

for k in range(n_b):
   for w in range((k*blocksize)+1,(k*blocksize)+blocksize+1):
      B += data[w-1]
   Data.insert(k,B)
   B = 0

''' Do the jackknife estimates '''
for i in range(n_b-1):
   data_jack.append((data_tot - Data[i]) / (n - blocksize))
   data_av = data_tot / n # Do the overall averages
   data_av = data_av
   data_jack_av = 0.; data_jack_err = 0.
for i in range(n_b-1):
   dR = data_jack[i]
   data_jack_av += dR
   data_jack_err += dR**2

data_jack_av /= n_b-1
data_jack_err /= n_b-1

data_jack_err = sqrt((n_b - 2) * abs(data_jack_err - data_jack_av**2))
print(" %8.7f " " %6.7f" " %6.2f" % (data_jack_av, data_jack_err, n_b))
```

SECTION G

Solutions to selected Exercises

★ Solution to Exercise :

We now show that $\det(V) = \prod_{i<j}(\lambda_i - \lambda_j)$ where V is:

$$V = \begin{pmatrix} 1 & \lambda_1 & \lambda_1^2 & \cdots & \lambda_1^{N-1} \\ 1 & \lambda_2 & \lambda_2^2 & \cdots & \lambda_2^{N-1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & \lambda_N & \lambda_N^2 & \cdots & \lambda_N^{N-1} \end{pmatrix} = \lambda_i^{j-1}$$

We first note that the determinant is unchanged if we make the change to all columns except the first given by:

$$\lambda_i^{j-1} \to \lambda_i^{j-1} - \lambda_1 \lambda_i^{j-2} \tag{G.1}$$

then we have,

$$\det(V') = \begin{vmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & \lambda_2 - \lambda_1 & \lambda_2(\lambda_2 - \lambda_1) & \cdots & \lambda_2^{N-2}(\lambda_2 - \lambda_1) \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & \lambda_N - \lambda_1 & \lambda_N(\lambda_N - \lambda_1) & \cdots & \lambda_N^{N-2}(\lambda_N - \lambda_1) \end{vmatrix} \tag{G.2}$$

By using Laplace Expansion formula for determinants, along the first row we find that $\det(V') = \det(V'')$ where V'' is:

$$\det(V'') = \begin{vmatrix} \lambda_2 - \lambda_1 & \cdots & \cdots & \lambda_2^{N-2}(\lambda_2 - \lambda_1) \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_N - \lambda_1 & & \cdots & \lambda_N^{N-2}(\lambda_N - \lambda_1) \end{vmatrix} \tag{G.3}$$

Taking the factors common in each row, we get:

$$\det(V) = \det(V'') = (\lambda_2 - \lambda_1) \cdots (\lambda_N - \lambda_1) \begin{vmatrix} 1 & \cdots & \cdots & \lambda_2^{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & & \cdots & \lambda_N^{N-2} \end{vmatrix} \tag{G.4}$$

If we iterate this with this smaller matrix and carry on, we will see that we will get:

$$\det(V) = \prod_{i<j}(\lambda_j - \lambda_i). \tag{G.5}$$

To define a Vandermonde matrix and compute determinant, we can execute following command in MATHEMATICA :

```
V = Table[Subscript[\[Alpha], i]^j, {i, 1, 5}, {j, 0, 4}];
Det@V // Simplify
```

★ Solution to Exercise ... and additional comments: The basic idea of the loop equations of matrix models is to capture the invariance of the model under field redefinitions. This is also sometimes known as 'Schwinger-Dyson (SD) equations'. One of the exercises in the main text was to derive these equations. Here, we will sketch a proof for the interested reader. We start by noting that the integral of the total derivative vanish and hence:

$$\sum_{i,j}\int dM \frac{\partial}{\partial M_{ij}}\left((M^k)_{ij}\ e^{-N\mathrm{Tr}V(M)}\right)=0, \tag{G.6}$$

By computing the derivatives and using large N factorization, we obtain:

$$\left\langle \mathrm{Tr}M^k V'(M)\right\rangle = \sum_{l=0}^{k-1}\langle \mathrm{Tr}M^l\rangle\langle \mathrm{Tr}M^{k-l-1}\rangle \tag{G.7}$$

In the steps above, we have used two identities:

$$\frac{\partial}{\partial M_{ij}}(M^k)_{ij} = \sum_{l=0}^{k-1}(M^l)_{ii}(M^{k-l-1})_{jj} \tag{G.8}$$

and,

$$\frac{\partial}{\partial M_{ij}}e^{-N\mathrm{Tr}V(M)} = -NV'(M)_{ji}\ e^{-N\mathrm{Tr}V(M)} \tag{G.9}$$

where V' denotes the derivative w.r.t to the matrix. However, these loop equations are not valid when the integration is over some other matrix ensembles (such as orthogonal/symplectic). It is better to start from the eigenvalue integral representation to consider general $\beta\in\mathbb{C}$. We can write moments as:

$$\langle \mathrm{Tr}M^k\rangle = \frac{1}{Z}\int \Delta(\lambda)^\beta d\lambda_1\cdots d\lambda_N\ \exp\left(-\frac{N\beta}{2}\sum_i V(\lambda_i)\right)\sum_{i=1}^N \lambda_i^k \tag{G.10}$$

It is easy to derive 'generalized loop equations' from here using the fact that inntegral of total derivatives vanishes. We obtain:

$$\left\langle \mathrm{Tr}M^k V'(M)\right\rangle + \underbrace{k\left(\frac{2}{\beta}-1\right)\mathrm{Tr}M^{k-1}}_{\text{zero for }\beta=2} = \sum_{l=0}^{k-1}\langle \mathrm{Tr}M^l\rangle\langle \mathrm{Tr}M^{k-l-1}\rangle \tag{G.11}$$

★ Solution to Exercise ...:

We consider $X_\mu \to (1+\epsilon)X_\mu + \mathcal{O}(\epsilon^2)$ and $\mathcal{D}X \to (1+\epsilon D(N^2-1))\mathcal{D}X$ with the path integral:

$$Z = \int \mathcal{D}X e^{-S} = \int \mathcal{D}X \exp\Big[-\frac{1}{4g^2}\mathrm{Tr}[X_\mu, X_\nu]^2\Big] \tag{G.12}$$

The transformation changes Z by:

$$Z = Z + \epsilon\Big\{D(N^2-1)Z - 4\langle S\rangle Z\Big\} = Z(1+\epsilon\Big\{D(N^2-1) - 4\langle S\rangle\Big\}) \tag{G.13}$$

If we demand that Z remains invariant, the term in the paranthesis should vanish and we get the desired result:

$$\frac{D}{4} = \frac{\langle S\rangle}{N^2-1} \tag{G.14}$$

★ Executing following command in MATHEMATICA will check that Wigner distribution is observed. The deviation from the semi-circle distribution cane be seen for small n. [11].

```
n = 1000;
scaledSpectrum=Flatten[RandomVariate[scaledSpectrum\[ScriptCapitalD][n], 100]];
Show[Histogram[scaledSpectrum, {0.05}, PDF, ChartStyle -> LightOrange],
Plot[PDF[WignerSemicircleDistribution[1], x],{x, -1.5, 1.5}, PlotLegends -> None,
PlotStyle -> ColorData[27, 1]], ImageSize -> Medium]
```

★ Solution to Exercise **XXX**:

Consider (2.24) with $k=1$,

$$\Big\langle \mathrm{Tr}\Big(M^2 + gM^4\Big)\Big\rangle = 1 \tag{G.15}$$

This then implies,

$$\mathrm{Tr}M^4 \equiv t_4 = \frac{1-\mathrm{Tr}M^2}{g} \equiv \frac{1-t_2}{g} \tag{G.16}$$

We can extend this to $\mathrm{Tr}M^6 \equiv t_6$ which can be obtained in terms of t_2 as:

$$t_6 = \frac{2t_2 - \frac{(1-t_2)}{g}}{g}. \tag{G.17}$$

★ Solution to Exercise 5:

[11] Please see https://www.wolfram.com/language/11/random-matrices for more

We now show that $\langle e^{-\Delta H} \rangle = 1$ The Hamiltonian of the system is defined as:

$$H(P, X) = \frac{1}{2}P^2 + S(X) \tag{G.18}$$

Areas of phase space is preserved i.e., $dPdX = dP'dX'$ and hence we have:

$$\begin{aligned} Z &= \int dP'dX'e^{-H'} \\ &= \int dPdXe^{-H}e^{H-H'} \end{aligned} \tag{G.19}$$

Dividing (G.19) by Z we get,

$$\langle e^{H-H'} \rangle = \langle e^{-\Delta H} \rangle = 1 \tag{G.20}$$

★ Solution to Exercise XX:

We need to modify the potential and the corresponding forces as discussed in Appendix ??. In addition to the `def potential(X)` and `def force(X)` given in Appendix E, we add the following lines

```
def potential(X):
     for i in range (NSCALAR):
          tmp += h_q * NCOL * np.trace(X[i] @ X[i]).real


def force(X):
     for i in range (NSCALAR):
          f_X[i] += 2.0 * h_q * NCOL*X[i]
```

References

[1] E. Wigner, "On the statistical distribution of the widths and spacings of nuclear resonance levels," 1951.

[2] R. U. Haq, A. Pandey, and O. Bohigas, "Fluctuation properties of nuclear energy levels: Do theory and experiment agree?," *Phys. Rev. Lett.* **48** (Apr, 1982) 1086–1089. https://link.aps.org/doi/10.1103/PhysRevLett.48.1086.

[3] M. Mehta, "On the statistical properties of the level-spacings in nuclear spectra," *Nuclear Physics* **18** (1960) 395–419. https://www.sciencedirect.com/science/article/pii/0029558260904132.

[4] M. Gaudin, "Sur la loi limite de l'espacement des valeurs propres d'une matrice aleÂtatoire," *Nuclear Physics* **25** (1961) 447–458. https://www.sciencedirect.com/science/article/pii/0029558261901766.

[5] M. L. Mehta, *Random Matrices*. 3rd ed., 2004.

[6] G. Akemann, J. Baik, and P. Di Francesco, *The Oxford Handbook of Random Matrix Theory*. Oxford Handbooks in Mathematics. Oxford University Press, 9, 2011.

[7] P. Di Francesco, P. H. Ginsparg, and J. Zinn-Justin, "2-D Gravity and random matrices," *Phys. Rept.* **254** (1995) 1–133, arXiv:hep-th/9306153.

[8] B. Eynard, T. Kimura, and S. Ribault, "Random matrices," arXiv:1510.04430 [math-ph].

[9] Y. Y. Goldschmidt, "$1/N$ Expansion in Two-dimensional Lattice Gauge Theory," *J. Math. Phys.* **21** (1980) 1842.

[10] E. Brezin, C. Itzykson, G. Parisi, and J. B. Zuber, "Planar Diagrams," *Commun. Math. Phys.* **59** (1978) 35.

[11] M. Marino, "Les Houches lectures on matrix models and topological strings," 10, 2004. arXiv:hep-th/0410165.

[12] A. A. Migdal, "Loop Equations and 1/N Expansion," *Phys. Rept.* **102** (1983) 199–290.

[13] S. Chadha, G. Mahoux, and M. L. Mehta, "A Method of Integration Over Matrix Variables. 2.," *J. Phys. A* **14** (1981) 579.

[14] P. D. Anderson and M. Kruczenski, "Loop Equations and bootstrap methods in the lattice," *Nucl. Phys. B* **921** (2017) 702–726, arXiv:1612.08140 [hep-th].

[15] H. W. Lin, "Bootstraps to strings: solving random matrix models with positivite," *JHEP* **06** (2020) 090, arXiv:2002.08387 [hep-th].

[16] X. Han, S. A. Hartnoll, and J. Kruthoff, "Bootstrapping Matrix Quantum Mechanics," *Phys. Rev. Lett.* **125** no. 4, (2020) 041601, arXiv:2004.10212 [hep-th].

[17] V. Kazakov and Z. Zheng, "Analytic and Numerical Bootstrap for One-Matrix Model and "Unsolvable" Two-Matrix Model," arXiv:2108.04830 [hep-th].

[18] M. Hanada, "Markov Chain Monte Carlo for Dummies," arXiv:1808.08490 [hep-th].

[19] M. Staudacher, "The Yang-lee Edge Singularity on a Dynamical Planar Random Surface," *Nucl. Phys. B* **336** (1990) 349.

[20] V. A. Kazakov, I. K. Kostov, and N. A. Nekrasov, "D particles, matrix integrals and KP hierarchy," *Nucl. Phys. B* **557** (1999) 413–442, arXiv:hep-th/9810035.

[21] W. Krauth and M. Staudacher, "Finite Yang-Mills integrals," *Phys. Lett. B* **435** (1998) 350–355, arXiv:hep-th/9804199.

[22] W. Krauth and M. Staudacher, "Eigenvalue distributions in Yang-Mills integrals," *Phys. Lett. B* **453** (1999) 253–257, arXiv:hep-th/9902113.

[23] T. Hotta, J. Nishimura, and A. Tsuchiya, "Dynamical aspects of large N reduced models," *Nucl. Phys. B* **545** (1999) 543–575, arXiv:hep-th/9811220.

[24] T. Banks, W. Fischler, S. H. Shenker, and L. Susskind, "M theory as a matrix model: A Conjecture," *Phys. Rev. D* **55** (1997) 5112–5128, arXiv:hep-th/9610043.

[25] S. Catterall, R. G. Jha, D. Schaich, and T. Wiseman, "Testing holography using lattice super-Yang-Mills theory on a 2-torus," *Phys. Rev. D* **97** no. 8, (2018) 086020, arXiv:1709.07025 [hep-th].

[26] R. G. Jha, S. Catterall, D. Schaich, and T. Wiseman, "Testing the holographic principle using lattice simulations," *EPJ Web Conf.* **175** (2018) 08004, arXiv:1710.06398 [hep-lat].

[27] S. Catterall, J. Giedt, R. G. Jha, D. Schaich, and T. Wiseman, "Three-dimensional super-Yang–Mills theory on the lattice and dual black branes," *Phys. Rev. D* **102** no. 10, (2020) 106009, arXiv:2010.00026 [hep-th].

[28] N. Ishibashi, H. Kawai, Y. Kitazawa, and A. Tsuchiya, "A Large N reduced model as superstring," *Nucl. Phys. B* **498** (1997) 467–491, arXiv:hep-th/9612115.

[29] F. R. Klinkhamer, "A first look at the bosonic master-field equation of the IIB matrix model," arXiv:2105.05831 [hep-th].

[30] P. Young, "Everything you wanted to know about Data Analysis and Fitting but were afraid to ask," *arXiv e-prints* (Oct., 2012) arXiv:1210.3781, arXiv:1210.3781 [physics.data-an].