A Users' Manual of Monte Carlo Code for the Matrix Model of M-theory (matrix v10.4)

Masanori HANADA[a,b,c,d1]

[a] *Stanford Institute for Theoretical Physics, Stanford University, Stanford, CA 94305, USA*
[b] *Yukawa Institute for Theoretical Physics, Kyoto University, Kitashirakawa Oiwakecho, Sakyo-ku, Kyoto 606-8502, Japan*
[c] *The Hakubi Center for Advanced Research, Kyoto University, Yoshida Ushinomiyacho, Sakyo-ku, Kyoto 606-8501, Japan*
[d] *Physics Division, Lawrence Livermore National Laboratory, Livermore CA 94550, USA*

abstract

This is a Users' manual of an open-source Monte Carlo simulation code for the Matrix Model of M-theory (BFSS matrix model) and its plane-wave deformation (BMN matrix model), written and managed by Masanori Hanada. There is no black box, everything is written explicitly by using Fortran 90 and MPI (Message Passing Interface). As you can immediately see, I do not use any fancy technique in the code; I just use a lot of *do* and *if*, and hence you can read it if you know high-school level math and the very basics of Fortran and MPI.[2]

IMPORTANT: If you find a bug, please let me know. Then I will buy you a glass of beer. The latest version is **matrix_v10.4**. Please do not use the OpenMP option. It may cause a bug.

[1] E-mail address : hanada@yukawa.kyoto-u.ac.jp

[2] In order to understand the algorithm, undergrad level math might be needed.

Contents

1 Introduction

This is a manual of a Monte Carlo simulation code for the Matrix Model of M-theory (BFSS matrix model) [1, 2] and its plane-wave deformation (BMN matrix model) [3]. In this simulation code, no special package is used and everything is written explicitly. The only thing you need is Fortran90 complier linked to MPI (Message Passing Interface). For an apparent reason, this manual should be called M5.

By using this simulation code, one can study BFSS [1, 2] and BMN [3] matrix models. BFSS matrix model is believed to describe super-membrane [2], M-theory[1] and type IIA superstring [4], depending on the parameter regions. BMN matrix model has various BPS vacua, and by choosing appropriate vacuum it is expected to describe D2-brane and M2-brane [5] (i.e. 3d SYM), NS5-brane and M5-brane [5] and planar 4d N=4 SYM [6].

Although you (the users) should be responsible for the simulation results obtained by using this code, when you find a bug, or have some requests for new features, please do not hesitate to contact me. I will be more than happy to help you.

1.1 Structure of this manual

This manual is organized as follows.

- The action is explained in Sec. 2. (The continuum theory is explained in Sec. 2.1. Two lattice regularizations (naive and improved) are explained in Sec. 2.3 and Sec. 2.4. In order to perform simulations efficiently, I modify the lattice action slightly, without

changing the result; see Sec. 2.5. In Sec. 2.5.4, an additional term, which is used to control the flat direction, is explained.

- The simulation algorithm is explained in Sec. 3. In this section, I assume the readers are usual string theorists, who do not have any knowledge on the lattice Monte Carlo simulation. However, even for lattice practitioners, this section should be useful in order to understand the detail of the algorithm employed in my code.

- The structure of the simulation code is explained in Sec. 4.

  Sec. 4.2 explains how to compile and run the code. Parameters needed for compilation and run are explained in Sec. 4.2 and Sec. 4.3, respectively.

  Sec. 4.4 explains how to read the measurement results. In this section, you can also learn how to choose appropriate input parameters by looking at output.

  In Sec. 4.5, the main routine is explained. There are several subroutines, in which various 'sub-subroutines,' which are explained in Sec. 4.6, are called.

Note that Sec. 2 and Sec. 3 are essentially copied and pasted from [7]. I repeat it here in order to make this manual self-contained. In future, when I improve the regularization or algorithm, I will update this manual for sure, but [7] will not necessarily be updated.

2 Action

2.1 Continuum Theory

2.1.1 BFSS Matrix Model

Let us consider the BFSS matrix model on Euclidean circle with the circumference β. For bosonic fields, I always impose the periodic boundary condition. When the boundary condition for the fermion is anti-periodic, β is the inverse of the temperature, $\beta = 1/T$.

This model consists of nine $N \times N$ bosonic hermitian matrices X_M ($M = 1, 2, \cdots, 9$), sixteen fermionic matrices ψ_α ($\alpha = 1, 2, \cdots, 16$) and the gauge field A_t. Both X_M and ψ_α are in the adjoint representation of $U(N)$ gauge group, and the covariant derivative D_t acts on them as $D_t X_M = \partial_t X_M - i[A_t, X_M]$ and $D_t \psi_\alpha = \partial_t \psi_\alpha - i[A_t, \psi_\alpha]$. The action is given by[3]

$$S_{BFSS} = S_b + S_f, \tag{1}$$

where S_b is the bosonic part,

$$S_b = N \int_0^\beta dt \ \mathrm{Tr} \left\{ \frac{1}{2} (D_t X_M)^2 - \frac{1}{4} [X_M, X_N]^2 \right\}, \tag{2}$$

[3]In this theory, the 't Hooft coupling $\lambda = g_{YM}^2 N$ has a dimension of (mass)^3, and can be set to 1 by proper rescaling of time t and matrices. In other words, all dimensionful quantities can be set dimensionless by multiplying appropriate powers of λ. In this code, $\lambda = 1$.

and S_f is the fermionic part,

$$S_f = N \int_0^\beta dt \ \mathrm{Tr} \left\{ i\bar{\psi}\gamma^{10} D_t \psi - \bar{\psi}\gamma^M [X_M, \psi] \right\}. \tag{3}$$

Here γ^M $(M = 1, \cdots, 10)$, which are 16×16 upper-right block of the 10d Gamma matrices Γ^M. For the latter convenience, I take[4]

$$\gamma_{10} = \sigma_1 \otimes \mathbf{1}_8. \tag{5}$$

This model is obtained by dimensionally reducing the ten-dimensional $\mathcal{N} = 1$ super Yang-Mills theory to one dimension. The index α of the fermionic matrices ψ_α corresponds to the spinor index in ten dimension, and ψ_α is Majorana-Weyl in ten-dimensional sense.

In order for numerical efficiency, we take the static diagonal gauge,

$$A_t = \frac{1}{\beta} \cdot \mathrm{diag}(\alpha_1, \cdots, \alpha_N), \qquad -\pi < \alpha_i \leq \pi. \tag{6}$$

Associated with this gauge fixing, we add the Faddeev-Popov term $-\sum_{i<j} 2 \log \left| \sin\left(\frac{\alpha_i - \alpha_j}{2}\right) \right|$ to the action.

$$S_{F.P.} \ = \ -\sum_{i<j} 2 \log \left| \sin\left(\frac{\alpha_i - \alpha_j}{2}\right) \right|, \tag{7}$$

2.1.2 BMN Matrix Model

The plane-wave deformed theory, which is called the BMN matrix model [3], is given by

$$S_{BMN} = S_{BFSS} + \Delta S, \tag{8}$$

[4]Others are taken as follows:

$$\begin{aligned}
\gamma^1 &= \sigma_3 \otimes \mathbf{1} \otimes \mathbf{1} \otimes \mathbf{1}, \\
\gamma^2 &= \sigma_2 \otimes \sigma_2 \otimes \sigma_2 \otimes \sigma_2, \\
\gamma^3 &= \sigma_2 \otimes \sigma_2 \otimes \mathbf{1} \otimes \sigma_1, \\
\gamma^4 &= \sigma_2 \otimes \sigma_2 \otimes \mathbf{1} \otimes \sigma_3, \\
\gamma^5 &= \sigma_2 \otimes \sigma_1 \otimes \sigma_2 \otimes \mathbf{1}, \\
\gamma^6 &= \sigma_2 \otimes \sigma_3 \otimes \sigma_2 \otimes \mathbf{1}, \\
\gamma^7 &= \sigma_2 \otimes \mathbf{1} \otimes \sigma_1 \otimes \sigma_2, \\
\gamma^8 &= \sigma_2 \otimes \mathbf{1} \otimes \sigma_3 \otimes \sigma_2, \\
\gamma^9 &= -i\mathbf{1} \otimes \mathbf{1} \otimes \mathbf{1} \otimes \mathbf{1}.
\end{aligned} \tag{4}$$

By taking $\Gamma^i = \sigma_1 \otimes \gamma^i$ $(i = 1, 2, \cdots, 8, 10)$ and $\Gamma^9 = i\sigma_2 \otimes \gamma^9$, the standard anticommutation relation holds, $\{\Gamma^M, \Gamma^N\} = 2\delta^{MN}$.

where

$$\Delta S_b = N \int_0^\beta dt \ \mathrm{Tr} \left\{ \frac{\mu^2}{2} \sum_{i=1}^{3} X_i^2 + \frac{\mu^2}{8} \sum_{a=4}^{9} X_a^2 + i \sum_{i,j,k=1}^{3} \mu \epsilon^{ijk} X_i X_j X_k \right\} \tag{9}$$

and

$$\Delta S_f = \frac{3i\mu}{4} \cdot N \int_0^\beta dt \ \mathrm{Tr} \left(\bar{\psi} \gamma^{123} \psi \right). \tag{10}$$

Here ϵ_{ijk} is the structure constant of $SU(2)$, and hence $i \sum_{i,j,k=1}^{3} \mu \epsilon^{ijk} \mathrm{Tr} \left(X_i X_j X_k \right) = 3i \mathrm{Tr} \left(X_1, [X_2, X_3] \right)$.

We take the static diagonal gauge. The Faddeev-Popov term is the same as in the BFSS matrix model, (7).

2.2 Apologia

For a stupid historical reason (a.k.a. a typo in my note), gamma matrices in my code, which I call $\tilde{\gamma}_M$, differs from γ_M by a factor i:

$$\tilde{\gamma}_M = -i \gamma_M. \tag{11}$$

(You can see that I multiplied $(-i)$ at the end of subroutine **MakeGamma**.) Furthermore, in my code, 'Dirac operator' is actually $(-2ia)$ times the Dirac operator, where a is the lattice spacing. I hope you not to be confused much.[5]

2.3 Naive Lattice Regularization

2.3.1 BFSS Matrix Model

We regularize the theory by introducing a lattice with N_t sites. Our lattice action is

$$S_b = \frac{N}{2a} \sum_{t,M} \mathrm{Tr} \left(U X_M(t+a) U^\dagger - X_M(t) \right)^2 - \frac{Na}{4} \sum_{t,M,N} \mathrm{Tr} [X_M(t), X_N(t)]^2, \tag{12}$$

$$S_{F.P.} = - \sum_{i<j} 2 \log \left| \sin \left(\frac{\alpha_i - \alpha_j}{2} \right) \right|, \tag{13}$$

$$S_f = iN \sum_t \mathrm{Tr} \bar{\psi}(t) \begin{pmatrix} 0 & D_+ \\ D_- & 0 \end{pmatrix} \psi(t) - aN \sum_{t,M} \bar{\psi}(t) \gamma^M [X_M(t), \psi(t)], \tag{14}$$

where $U = \mathrm{diag}(e^{i\alpha_1/N_t}, e^{i\alpha_2/N_t} \cdots, e^{i\alpha_N/N_t})$, $-\pi \le \alpha_i < \pi$, and

$$\begin{aligned} D_+ \psi(t) &= U \psi(t+a) U^\dagger - \psi(t), \\ D_- \psi(t) &= \psi(t) - U^\dagger \psi(t-a) U. \end{aligned} \tag{15}$$

Other than the gauge fixing, this action is the same as the one used in [16].

[5] I believe this notation is not as annoying as $(+,-,-,-)$.

2.3.2 BMN Matrix Model

The plane wave deformation is

$$\Delta S_b = aN \sum_t \ \mathrm{Tr} \left\{ \frac{\mu^2}{2} \sum_{i=1}^{3} X_i(t)^2 + \frac{\mu^2}{8} \sum_{a=4}^{9} X_a(t)^2 + i \sum_{i,j,k=1}^{3} \mu \epsilon^{ijk} X_i(t) X_j(t) X_k(t) \right\} \quad (16)$$

and

$$\Delta S_f = \frac{3i\mu}{4} \cdot aN \sum_t \ \mathrm{Tr} \left(\bar{\psi}(t) \gamma^{123} \psi(t) \right). \quad (17)$$

2.4 Tree-level Improved Lattice Regularization

In the naive lattice discretization,

$$D_\pm \psi(t) = a D_t \psi(t) \pm \frac{a^2}{2} D_t^2 \psi(t) + O(a^3), \quad (18)$$

where D_t is the covariant derivative. We can reduce the discretization error by using

$$\tilde{D}_\pm \psi(t) \equiv - \mp \frac{1}{2} U^2 \psi(t \pm 2a) \left(U^\dagger \right)^2 \pm 2U \psi(t \pm a) U^\dagger \mp \frac{3}{2} \psi(t) = a D_t \psi(t) + O(a^3). \quad (19)$$

The improved action is

$$S_{f,improved} = iN \sum_t \mathrm{Tr} \bar{\psi}(t) \begin{pmatrix} 0 & \tilde{D}_+ \\ \tilde{D}_- & 0 \end{pmatrix} \psi(t) - aN \sum_{t,M} \bar{\psi}(t) \Gamma^M [X_M(t), \psi(t)], \quad (20)$$

$$S_{b,improved} \ = \ \frac{N}{2a} \sum_{t,M} \mathrm{Tr} \left(\tilde{D}_+ X_M(t) \right)^2 - \frac{Na}{4} \sum_{t,M,N} \mathrm{Tr} [X_M(t), X_N(t)]^2. \quad (21)$$

The plane-wave deformation and the Faddeev-Popov term remains unchanged.

2.5 Some Technicalities in Actual Numerical Simulation

2.5.1 Projection of the trace part of ψ

Small eigenvalues in the Dirac operator makes the simulation more costly. The $U(1)$ part is particularly problematic, because small eigenvalues arising from this sector are not lifted by the interaction, and even worse, it provides us with the exact zero mode when we use the periodic boundary condition. However, the $U(1)$ part is just a free theory which is decoupled from other sectors; we can throw it away without affecting the dynamics. For this reason, we impose the traceless condition on the fermion ψ,

$$\sum_{i=1}^{N} \psi_{ii}(t) = 0. \quad (22)$$

The 'pseudo-fermion,' which is introduced later for the RHMC algorithm, is also made traceless, associated with the tracelessness of ψ.

2.5.2 Projection of the trace part of $\int dt X_M(t)$

For the scalar fields, we impose

$$\int dt \sum_{i=1}^{N} X_{M,ii}(t) = 0, \tag{23}$$

in order to suppress the random walk of the center of mass.

Note that we can can impose these traceless conditions both in the BFSS and BMN matrix models.

2.5.3 Constraint for α_i

As I have explained before, the Polyakov line phases are constrained to $-\pi < \alpha_i \leq \pi$. If we just impose this constraint as it is, then the simulation is not very efficient – the 'center of mass' $(\sum_i \alpha_i)/N$ randomly walks and hits $\pm\pi$. So, we use the following trick.

Firstly, let us remind that α_i and $\tilde{\alpha}_i \equiv \alpha_i - \frac{\sum_j \alpha_j}{N}$ give the same weight. Equivalently, once $\tilde{\alpha}_i$ with the conditions $\sum_i \tilde{\alpha}_i = 0$ and $\max(\tilde{\alpha}_i) - \min(\tilde{\alpha}_i) < 2\pi$ are given, we can construct α_i by adding a constant to $\tilde{\alpha}_i$ as $\alpha_i = \tilde{\alpha}_i + C$. Here, C must satisfy $\min(\tilde{\alpha}_i) + C > -\pi$ and $\max(\tilde{\alpha}_i) + C \leq \pi$, because of the condition $-\pi < \alpha_i \leq \pi$. Hence C must sit in $(-\pi - \min(\tilde{\alpha}_i), \pi - \max(\tilde{\alpha}_i)]$, whose interval is $2\pi - \mu$, where $\mu \equiv \max(\tilde{\alpha}_i) - \min(\tilde{\alpha}_i)$. Therefore, we can replace the integral over α_i with that over $\tilde{\alpha}_i$, which satisfies the condition $\sum_i \tilde{\alpha}_i = 0$, with an additional Boltzmann weight

$$w(\mu) = \begin{cases} 2\pi - \mu & (\mu < 2\pi) \\ 0 & (\mu \geq 2\pi). \end{cases} \tag{24}$$

Projection of the $U(1)$ part is done every time a new configuration is accepted. (It is easy to see that we don't have to keep the tracelessness during the molecular evolution.)

2.5.4 A cutoff for TrX_M^2/N

In the BFSS matrix model, there is a flat direction $[X_M, X_{M'}] = 0$. It can also be a problem with the BMN matrix model, when the flux parameter μ is small. In order to tame this flat direction, we add the following potential[6]:

$$S_R = \begin{cases} 0 & (\frac{1}{N\beta}\int dt \mathrm{Tr} X_M^2 < R) \\ g_R \cdot N \cdot \left(\frac{1}{N\beta}\int dt \mathrm{Tr} X_M^2 - R \right) & (\frac{1}{N\beta}\int dt \mathrm{Tr} X_M^2 \geq R). \end{cases} \tag{25}$$

By taking R and g_R appropriately, we can cut off the flat direction.

Different from the modifications explained in Sec. 2.5, this cutoff potential can change the theory. For example, when you study the trivial vacuum (i.e. eigenvalues of X_M are

[6]For a historical reason, I use R, rather than R^2, to denote the cutoff for $\frac{1}{N\beta}\int dt \mathrm{Tr} X_M^2$. It might be a bad notation, I hope you will not be confused much.

bounded near the origin), if $\frac{1}{N\beta}\int dt \mathrm{Tr} X_M^2$ only rarely reaches R (say once in 1000 trajectories), one can use the simulation result as a good approximation of the trivial vacuum. On the other hand, if $\frac{1}{N\beta}\int dt \mathrm{Tr} X_M^2$ hits R often, it cannot be a good approximation, unless one uses more sophisticated configuration-selection procedure.

2.6 Apologia 2: lattice vs. momentum cutoff method

Previously, I and collaborators advertised the momentum cutoff method ('non-lattice method') [15] and did use it in various projects. Then why am I using lattice now? The reason is that, while the momentum cutoff method has various advantages such as a fast convergence to the continuum limit and quick restoration of supersymmetry, it is not adequate for a large-scale parallelization, because the action is highly nonlocal in momentum space. This simulation code is aimed for the study of very large N and/or low temperature, to which the large-scale parallelization is inevitably needed. Still we keep one of the advantages of the momentum cutoff method: the Fourier acceleration.

If you study SUSY quantum mechanics at small scale, I recommend you to use the momentum cutoff method.

2.7 Smearing scalars in Dirac operator (matrix v10.2 and later)

In order to improve Dirac operator, we smear the scalars in Dirac operator. Let us define a one-smearing procedure by

$$f(X_M)(t) = (1-2s)X_M(t) + s\left(X_M(t+a) + X_M(t-a)\right). \tag{26}$$

Here s is a free parameter, whose natural value is between $\frac{1}{3}$ and 0. Note that we do not need to care about the gauge invariance, we have already fixed the gauge!

We perform the smearing n_{smear} times. By using the notation $f(f(x)) = f^2(x), f(f^2(x)) = f^3(x)$ etc,

$$X_M^{\mathrm{smeared}}(t) = f^{n_{\mathrm{smear}}}(X_M)(t). \tag{27}$$

Note that we smear the scalars in the Dirac operator only; if we modify the bosonic part as well, then it is just a change of variables.

This can be implemented to our simulation code very easily:

- Prepare a subroutine for the smearing (27).
- Whenever the Dirac operator is used, replace X with X^{smeared}.

The force $\frac{\partial S}{\partial X}$ can be calculated by using the chain rule:

$$\frac{\partial S_F[f(X)]}{\partial X(t)} = \frac{\partial f(X)(t')}{\partial X(t)}\frac{\partial S_F[f(X)]}{\partial f(X)(t')} = (1-2s)\frac{\partial S_F[f(X)]}{\partial f(X)(t)} + s\left(\frac{\partial S_F[f(X)]}{\partial f(X)(t+a)} + \frac{\partial S_F[f(X)]}{\partial f(X)(t-a)}\right).$$

(28)

In order to simplify the coding, let us use f for generic function,

$$f\left(h(t)\right) = (1-2s)h(t) + s\left(h(t+a) + h(t-a)\right). \tag{29}$$

Then

$$f\left(\frac{\partial S_F[X]}{\partial X(t)}\right) = (1-2s)\frac{\partial S_F[X]}{\partial X(t)} + s\left(\frac{\partial S_F[X]}{\partial X(t+a)} + \frac{\partial S_F[X]}{\partial X(t-a)}\right), \tag{30}$$

and hence

$$\frac{\partial S_F[f^{n+1}(X)]}{\partial f^n(X)(t)} = f\left(\frac{\partial S_F[f^{n+1}(X)]}{\partial f^{n+1}(X)(t)}\right), \tag{31}$$

$$\begin{aligned}
\frac{\partial S_F[f^{n+2}(X)]}{\partial f^n(X)(t)} &= \frac{\partial f^{n+1}(X)(t')}{\partial f^n(X)(t)}\frac{\partial S_F[f^{n+2}(X)]}{\partial f^{n+1}(X)(t')} \\
&= (1-2s)f\left(\frac{\partial S_F[f^{n+2}(X)]}{\partial f^{n+2}(X)(t)}\right) + s\left(f\left(\frac{\partial S_F[f^{n+2}(X)]}{\partial f^{n+2}(X)(t+a)}\right) + f\left(\frac{\partial S_F[f^{n+2}(X)]}{\partial f^{n+2}(X)(t-a)}\right)\right) \\
&= f^2\left(\frac{\partial S_F[f^{n+2}(X)]}{\partial f^{n+2}(X)(t)}\right)
\end{aligned} \tag{32}$$

and

$$\frac{\partial S_F[f^n(X)]}{\partial X(t)} = f^n\left(\frac{\partial S_F[f^n(X)]}{\partial f^n(X)(t)}\right). \tag{33}$$

Sadly, the smearing does not give us any gain. So we don't need to use it.

3 Simulation method

Suppose that a positive (semi-)definite path-integral weight $P[X_M, \alpha]$ is given. In the Markov-chain Monte Carlo method, a chain of sets of field configurations $\{X_M^{(k)}, \alpha^{(k)}\}$ $(k = 1, 2, 3, \cdots)$ is generated so that the probability of obtaining a configuration $\{X_M, \alpha\}$ converges to $P[X_M, \alpha]$. In the BFSS and BMN matrix models, the path-integral weight is complex, and hence we have to rewrite the problem so that the Monte Carlo method is used. We explain this in Sec. 3.1. Then we explain the Rational Hybrid Monte Carlo (RHMC) algorithm, which is implemented in this simulation code, in Sec. 3.2.

3.1 Phase-quench approximation

The fermionic action S_f can be written as $S_f = \frac{1}{2}\mathcal{M}_{A\alpha r;B\beta s}\tilde{\psi}^A_{\alpha r}\tilde{\psi}^B_{\beta s}$, where we have expanded $\tilde{\psi}_{\alpha r}$ in terms of $U(N)$ generators t^A, $\tilde{\psi}_{\alpha r} = \sum_{A=1}^{N^2}\tilde{\psi}^A_{\alpha r}t^A$. By integrating out the fermionic variables, the path-integral measure becomes

$$\int [dX][d\alpha][d\psi]e^{-S_b - S_f} \;\to\; \int [dX][d\alpha][d\psi](\mathrm{Pf}\mathcal{M})e^{-S_b}, \tag{34}$$

where Pf$\mathcal{M}$ is the Pfaffian of $\mathcal{M}$, which is complex in general[7]. Because the Monte-Carlo simulation is applicable only when the path-integral measure is positive, we replace the Pfaffian with its absolute value, that is, we calculate

$$\langle \mathcal{O}(X,\alpha)\rangle \equiv \frac{\int [dX][d\alpha][d\psi]|\mathrm{Pf}\mathcal{M}|e^{-S_b}\cdot \mathcal{O}(X,\alpha)}{\int [dX][d\alpha][d\psi]|\mathrm{Pf}\mathcal{M}|e^{-S_b}}. \tag{35}$$

In general, such 'approximation' changes the theory and hence the expectation values. The expectation value in the original theory can be calculated in principle as

$$\langle \mathcal{O}(X,\alpha)\rangle_{original} \equiv \frac{\langle \mathcal{O}(X,\alpha)\cdot \mathrm{Pf}\mathcal{M}/|\mathrm{Pf}\mathcal{M}|\rangle}{\langle \mathrm{Pf}\mathcal{M}/|\mathrm{Pf}\mathcal{M}|\rangle}. \tag{36}$$

However there are several reasons we do not want to use (36). Firstly, calculation of the pfaffian is numerically very demanding, and although we can avoid explicit evaluation of the Pfaffian in (35) (we will explain this point later), in order to calculate (36) we must calculate the Pfaffian explicitly. Secondly, if the phase factor Pf$\mathcal{M}$/|Pf$\mathcal{M}$| fluctuates rapidly, both the denominator and numerator becomes very small, and then numerically it is difficult to distinguish them from zero. Then the expression is practically 0/0, whose error bar is infinitely large.

Surprisingly, and very fortunately, this 'approximation' turns out to be good in the present case. For example, in the finite-temperature simulation at $T \geq 0.45$ and $N \leq 17$ performed in [13, 14] the phase factor is very close to 1 and hence it can safely be ignored. At low temperature, the phase factor does oscillate. Still, as long as the fluctuation is not very rapid, we can evaluate $\langle \mathcal{O}(X,\alpha)\rangle_{original}$ by using (36), and then it agrees with $\langle \mathcal{O}(X,\alpha)\rangle$ within numerical error. When the phase factor fluctuates rapidly, we cannot evaluate $\langle \mathcal{O}(X,\alpha)\rangle_{original}$ by using (36). Still, $\langle \mathcal{O}(X,\alpha)\rangle$ agrees to the dual gravity prediction, in all previous calculations.

3.2 Rational Hybrid Monte Carlo (RHMC) algorithm

Suppose a path-integral weight $e^{-S(x_1,x_2,\cdots,x_p)}$ is given, as a function of finitely many variables $(x_1, x_2, \cdots, x_p)$. (In our case, they corresponds to X_M and α on a lattice.) We assume S is real and the partition function $Z = \int dx_1 \cdots dx_p e^{-S(x_1,x_2,\cdots,x_p)}$ is finite. In the Monte Carlo simulation[8], we construct a chain of sets of variables $\{x^{(1)}\} \to \{x^{(2)}\} \to \cdots \{x^{(k)}\} \to \{x^{(k+1)}\} \to \cdots$, satisfying the following conditions.

- Markov Chain. –

  The probability of obtaining $\{x^{(k+1)}\}$ from $\{x^{(k)}\}$ does not depend on the previous configurations $\{x^{(1)}\}, \{x^{(2)}\}, \cdots, \{x^{(k-1)}\}$. We denote this probability by $P[\{x^{(k)}\} \to \{x^{(k+1)}\}]$.

[7]The Pfaffian, rather than the determinant, appears because ψ is Majorana.

[8]Strictly speaking, we should call it the Markov Chain Monte Carlo simulation, which is a special class of various Monte Carlo methods.

- Irreducibility. –

  Any two configurations are connected by finite steps.

- Aperiodicity. –

  The *period* a configuration $\{x\}$ is given by the greatest common divisor of possible numbers of steps to come back to itself. When the period is 1 for all configurations, the Markov chain is called aperiodic.

- Detailed balance condition. –

  The transition probability P satisfies $e^{-S[\{x\}]}P[\{x\} \to \{x'\}] = e^{-S[\{x'\}]}P[\{x'\} \to \{x\}]$.

Then, the probability distribution of $\{x^{(k)}\}(k = 1, 2, \cdots)$ converges to $P(x_1, x_2, \cdots, x_p) = e^{-S(x_1,x_2,\cdots,x_p)}/Z$ as the chain becomes longer. The expectation values are obtained by taking the average over the configurations,

$$\langle \hat{O} \rangle = \int dx_1 \cdots dx_p O(x_1, \cdots, x_p) P(x_1, x_2, \cdots, x_p) = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} O(x_1^{(k)}, \cdots, x_p^{(k)}). \quad (37)$$

This method is often called 'importance sampling,' because important field configurations are sampled more. In the path-integral of the lattice gauge theory, the phase space is huge (in d-dimensional SU(N) pure Yang-Mills theory, if we take L lattice sites along each directions, we have dL^d unitary link variables, each consisting of $N^2 - 1$ degrees of freedom, and hence there are $d(N^2-1)L^d$ integration variables), and hence it is practically impossible to perform the integral. Still, most of the phase space is not important (i.e. e^{-S} is extremely small) and by concentrating on important configurations it is possible to save the calculation cost.

3.2.1 Metropolis algorithm

As the simplest example, let us consider the weight $e^{-f(x)}$, where $f(x)$ is a real positive function of $x \in \mathbb{R}$. The Metropolis algorithm is as follows.

- Randomly choose $\Delta x \in \mathbb{R}$, and shift $x^{(k)}$ as $x^{(k)} \to x' \equiv x^{(k)} + \Delta x$. (Here Δx and $-\Delta x$ must appear with the same probability, so that the detailed balance condition is satisfied. For example we use the Gaussian random number, or the uniform random number between $\pm c$, where $c > 0$ is the 'step size.')

- Metropolis test: Generate a uniform random number r between 0 and 1. If $r < e^{f(x^{(k)})-f(x')}$, $x^{(k+1)} = x'$, i.e. the new value is 'accepted.' Otherwise $x^{(k+1)} = x^{(k)}$, i.e. the new configuration is 'rejected.'

It is an easy exercise to see that all conditions explained above, such as the detailed balance condition, are satisfied.

Let us see how it works, by using the Gaussian integral, $f(x) = e^{-x^2/2}$. We take $x^{(1)} = 0$, and Δx to be uniform random number between -0.5 and 0.5. In Fig. 1, we show the distribution of $x^{(1)}, x^{(2)}, \cdots, x^{(n)}$, for $n = 100, 1000, \cdots, 10,000,000$. We can see that the distribution converges to $e^{-x^2/2}/\sqrt{2\pi}$.

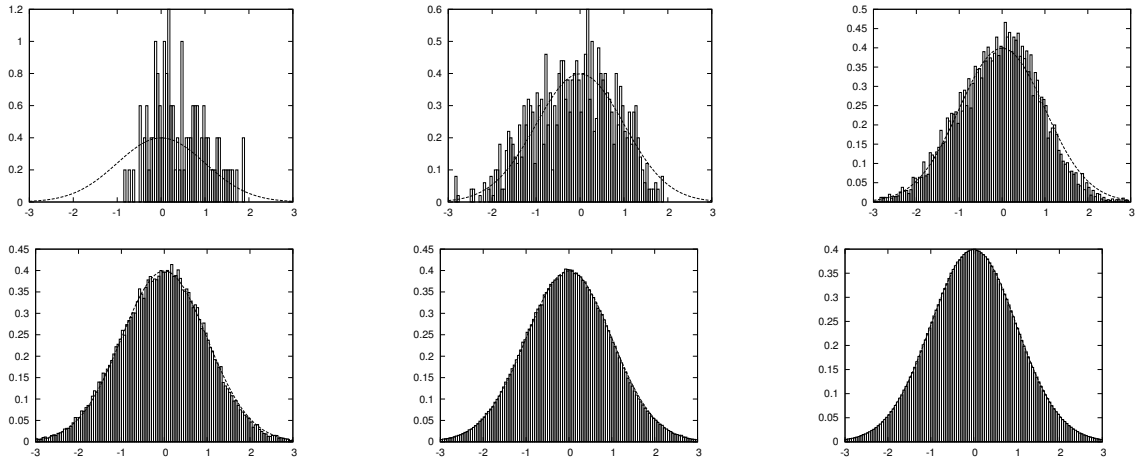


Figure 1: The distribution of $x^{(1)}, x^{(2)}, \cdots, x^{(n)}$, for $n = 100, 1000, \cdots, 10,000,000$. We used the Metropolis algorithm, as explained in the main text.

The Metropolis algorithm applies to the phase-quenched BFSS and BMN straightforwardly; we just have to vary multiple variables. Although this algorithm is very simple and can be applicable to any complicated system, however, there are several practical problems. The biggest problem is the long auto-correlation. If we vary the field too much (i.e. if c is large), then rather generically the action increases a lot and new configurations are almost always rejected. We can increase the acceptance rate by making the step size c smaller, but then it takes a lot of steps for the field configurations to change substantially. In any case, it takes long time to swipe the space of important configurations.

3.2.2 Hybrid Monte Carlo (HMC) algorithm

The Hybrid Monte Carlo (HMC) algorithm avoids the problem of a long auto-correlation. This is a 'hybrid' of molecular dynamical method and Metropolis algorithm.

In HMC algorithm, one generates sets of configurations $\{X_M^{(k)}, \alpha^{(k)}\}$ $(k = 1, 2, 3, \cdots)$ in the following manner.

Firstly, $\{X_M^{(1)}, \alpha^{(1)}\}$ can be arbitrary. Once $\{X_M^{(k)}, \alpha^{(k)}\}$ is obtained, $\{X_M^{(k+1)}, \alpha^{(k+1)}\}$ is obtained as follows.

- Randomly generate auxiliary momenta $P_X^{(k)}$ (hermitian) and $P_\alpha^{(k)}$ (real), which are 'conjugate' to X and α, with probabilities proportional to $e^{-Tr(P_X^{(k)})^2/2}$ and $e^{-\sum_i (P_{\alpha_i}^{(k)})^2/2}$.
- Calculate the 'Hamiltonian' $H_i = S[X^{(k)}, \alpha^{(k)}] + Tr(P_X^{(k)})^2/2 + \sum_i (P_{\alpha_i}^{(k)})^2/2$. Here S is the full effective action, $S = S_b - \log|\mathrm{Pf}\mathcal{M}|$.

- Then we consider 'time evolution' along an auxiliary time (which is not the Euclidean time!) τ. We set the initial condition to be $X^{(k)}(\tau = 0) = X^{(k)}$, $\alpha^{(k)}(\tau = 0) = \alpha^{(k)}$, $P_X^{(k)}(\tau = 0) = P_X^{(k)}$ and $P_{\alpha_i}^{(k)}(\tau = 0) = P_{\alpha_i}^{(k)}$, and use the leap frog method (see below) to calculate $X^{(k)}(\tau_f)$, $\alpha^{(k)}(\tau_f)$, $P_X^{(k)}(\tau_f)$ and $P_{\alpha_i}^{(k)}(\tau_f)$, where τ_f is related to the input parameters $\Delta\tau$ and N_τ by $\tau_f = N_\tau \Delta\tau$.

  This process is called 'molecular evolution.'

- Calculate $H_f = S[X^{(k)}(\tau_f), \alpha^{(k)}(\tau_f)] + Tr(P_X^{(k)}(\tau_f))^2/2 + \sum_i (P_{\alpha_i}^{(k)}(\tau_f))^2/2$.

- Metropolis test: Generate a uniform random number r between 0 and 1. If $r < e^{H_i - H_f}$, $X^{(k+1)} = X^{(k)}(\tau_f)$, $\alpha^{(k+1)} = \alpha^{(k)}(\tau_f)$, i.e. the new configuration is 'accepted.' Otherwise $X^{(k+1)} = X^{(k)}$, $\alpha^{(k+1)} = \alpha^{(k)}$, i.e. the new configuration is 'rejected.'

If we keep $N_\tau \Delta\tau$ fixed and send N_τ to infinity, then the Hamiltonian is exactly conserved and new configurations are always accepted. By taking $N_\tau \Delta\tau$ to be large, new configurations can be substantially different from the old ones. Of course, calculation cost increase with N_τ. So we have to find a sweet spot, with moderately large N_t and moderately small $\Delta\tau$.

In fact the HMC algorithm works even if we take different τ for α and X_M. We can even different τ depending the momentum of the mode. This fact leads to the Fourier acceleration, which is explained in Sec. 3.5.

Computationally the most costly parts are the calculations of $\log|\mathrm{Pf}\mathcal{M}|$ and its derivative. The RHMC algorithm avoids them, so that the cost is reduced drastically[9].

Leap frog method

The leap frog method is a clever way to discretize the continuum Hamiltonian equation keeping the reversibility, which is crucial for assuring the detailed balance condition.

The continuum Hamiltonian equation is given by

$$\frac{dp_{X_{Mij}}}{d\tau} = -\frac{\partial S}{\partial X_{Mji}} \equiv F_{X,Mij}, \qquad \frac{dX_{Mij}}{d\tau} = p_{X_{Mij}}, \tag{38}$$

$$\frac{dp_{\alpha_i}}{d\tau} = -\frac{\partial S}{\partial \alpha_i} \equiv F_{\alpha,i}, \qquad \frac{d\alpha_i}{d\tau} = p_{\alpha_i}. \tag{39}$$

The leap frog method goes as follows:

- $X_{Mij}(\Delta\tau_X/2) = X_{Mij}(0) + p_{X_{Mij}}(0) \cdot \Delta\tau/2$,
  $\alpha_i(\Delta\tau_\alpha/2) = \alpha_i(0) + p_{\alpha_i}(0) \cdot \Delta\tau/2$.

[9]For the two-flavor QCD with degenerate quark mass, we can avoid this problem without using the RHMC algorithm.

- For $n = 1, 2, N_\tau - 1$, repeat it:
  $p_{X,Mij}(n\Delta\tau) = p_{X,Mij}((n-1)\Delta\tau) + F_{X,Mij}((n-1/2)\Delta\tau)\cdot\Delta\tau$,
  $p_{\alpha,i}(n\Delta\tau) = p_{\alpha,i}((n-1)\Delta\tau) + F_{\alpha,i}((n-1/2)\Delta\tau)\cdot\Delta\tau$; then
  $X_{Mij}((n+1/2)\Delta\tau) = X_{Mij}((n-1/2)\Delta\tau) + p_{X_{Mij}}(n\Delta\tau)\cdot\Delta\tau$,
  $\alpha_i((n+1/2)\Delta\tau) = \alpha_i((n-1/2)\Delta\tau) + p_{\alpha_i}(n\Delta\tau)\cdot\Delta\tau$.

- Finally,
  $p_{X,Mij}(N_\tau\Delta\tau) = p_{X,Mij}((N_\tau-1)\Delta\tau) + F_{X,Mij}((N_\tau-1/2)\Delta\tau)\cdot\Delta\tau$,
  $p_{\alpha,i}(N_\tau\Delta\tau) = p_{\alpha,i}((N_\tau-1)\Delta\tau) + F_{\alpha,i}((N_\tau-1/2)\Delta\tau)\cdot\Delta\tau$; then
  $X_{Mij}(N_\tau\Delta\tau) = X_{Mij}((N_\tau-1/2)\Delta\tau) + p_{X_{Mij}}(N_\tau\Delta\tau)\cdot\Delta\tau/2$,
  $\alpha_i(N_\tau\Delta\tau) = \alpha_i((N_\tau-1/2)\Delta\tau) + p_{\alpha_i}(N_\tau\Delta\tau)\cdot\Delta\tau/2$.

For a visual-summary, see Fig. 2; X_M and α go 1/2 steps forward, then $p_{X,M}$ and p_α go one step forward, then X_M and α go one step forward, blah blah blah.

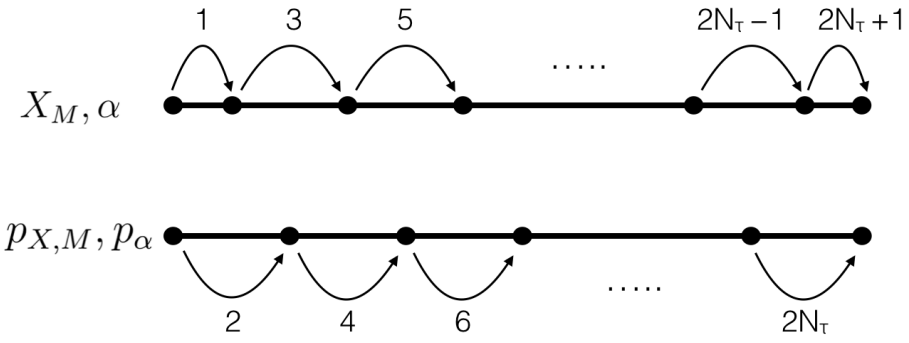


Figure 2: The leap-frog method.

3.2.3 RHMC = HMC + rational approximation

In order to evaluate (35) without calculating the Pfaffian and its derivative explicitly, we use the RHMC algorithm [10]. The starting point is to rewrite $|\mathrm{Pf}\mathcal{M}| = \left(\det(\mathcal{M}^\dagger\mathcal{M})\right)^{1/4}$ as

$$|\mathrm{Pf}\mathcal{M}| = \int dF dF^* \exp\left(-(D^{-1/8}F)^\dagger(D^{-1/8}F)\right), \tag{40}$$

where $D = \mathcal{M}^\dagger\mathcal{M}$, F is a complex bosonic field ('pseudo-fermion'), and dF is a usual flat measure. If we define Φ by $\Phi = D^{-1/8}F$, then Φ can be generated by the Gaussian weight

$e^{-\Phi^\dagger\Phi}$, and F can be obtained by solving $F = D^{1/8}\Phi$. In order to calculate F explicitly, we can use the rational approximation

$$x^{+1/8} \simeq a'_0 + \sum_{k=1}^{Q'} \frac{a'_k}{x + b'_k}. \tag{41}$$

Then

$$F \simeq a'_0\Phi + a'_k\chi'_k, \tag{42}$$

where

$$\frac{1}{D + b'_k}\Phi = \chi'_k \tag{43}$$

is obtained by solving

$$(D + b'_k)\chi'_k = \Phi. \tag{44}$$

We further rewrite this expression by using the rational approximation

$$x^{-1/4} \simeq a_0 + \sum_{k=1}^{Q} \frac{a_k}{x + b_k}. \tag{45}$$

Then we can replace the Pfaffian by

$$|\mathrm{Pf}\mathcal{M}| = \int dF dF^* \exp\left(-S_{\mathrm{PF}}\right), \tag{46}$$

where

$$S_{\mathrm{PF}} = a_0 F^\dagger F + \sum_{k=1}^{Q} a_k F^\dagger (D + b_k)^{-1} F. \tag{47}$$

The parameters a_k, b_k, a'_k and b'_k are real and positive, and can be chosen so that the approximation is sufficiently good within the range of eigenvalues of D during the simulation. (They can be obtained by a code [8] based on the Remez algorithm.)

The remaining part is just the same as HMC. Therefore, molecular evolution goes as follows:

- First we generate Φ by the Gaussian weight. (**Generate_pseudo_fermion_SUN**)
- Then we calculate F from Φ. (**Generate_pseudo_fermion_SUN**)
- Next we fix F and update X_M and α_i. This part is the usual molecular evolution. (**Molecular_Dynamics**)

Note that the conjugate momenta are introduced only for X_M and α_i.

Computationally most demanding part is solving the linear equations

$$(D + b_k)\chi_k = F \quad (k = 1, \cdots, Q), \tag{48}$$

which appears in the derivative of S_{PF},

$$\frac{\partial S_{PF}}{\partial X} = -\sum_{k=1}^{Q} a_k \chi_k^\dagger \frac{\partial D}{\partial X}\chi_k, \qquad \frac{\partial S_{PF}}{\partial \alpha} = -\sum_{k=1}^{Q} a_k \chi_k^\dagger \frac{\partial D}{\partial \alpha}\chi_k. \tag{49}$$

This is much easier than evaluating the Pfaffian. By using the idea of the multi-mass solver [9], all χ_k's can be obtained simultaneously by solving the equation for the smallest b_k. (See Appendix A for details.)

3.2.4 Reducing the condition number

We want to reduce the condition number of D so that the simulation becomes faster. We introduce a constant matrix K which is fixed during each run, and replace D with $\tilde{D} = K^{-1}DK^{-1}$. We write K in the momentum space, $K_{x,y} = C^{-1}_{x,p}\tilde{K}_{pq}C_{q,y}$. (We take K to be diagonal with respect to the spinor and gauge indices.) Here C and C^{-1} are the matrix representations of the Fourier transformation. Furthermore we take $\tilde{K}$ to be diagonal, $\tilde{K}_{pq} = \tilde{K}(p)\delta_{pq}$. Then $K^{-1}_{x,y} = C^{-1}_{x,p}\frac{1}{\tilde{K}(p)}C_{p,y}$. At UV, $\tilde{K}(p) \sim p$ is an appropriate choice. Another reasonable choice would be as follows. Firstly we prepare ϕ with a Gaussian random weight, then multiply D, and then Fourier transform it to the momentum space. We call it $\tilde{\phi}$: $\tilde{\phi}_p = C_{px}D_{xy}\phi_y$. (Modulo a certain normalization factor) $\tilde{K}(p) = \sqrt{\sum |\tilde{\phi}_p|^2}$. In practice, we can measure $\sqrt{\left\langle \sum |\tilde{\phi}_p|^2 \right\rangle}$ at an early run an use the same value for the data taking. (Or we can calculate it at the beginning of each run – right? We can use different K in each run.)

3.3 Traceless condition

Because we do not let the pseudo fermion evolve, the projection to $SU(N)$ can be achieved just by taking Φ to be traceless.

For X_M, we impose the traceless condition of the zero mode $\int dt Tr X_M = 0$ after the molecular evolution.

3.4 Constraint for α_i

As we explained previously, we rewrite the integral over α_i to that over $\tilde{\alpha}_i$, which satisfies the condition $\sum_i \tilde{\alpha}_i = 0$, with an additional Boltzmann weight

$$w(\mu) = \begin{cases} 2\pi - \mu & (\mu < 2\pi) \\ 0 & (\mu \geq 2\pi). \end{cases} \tag{50}$$

For numerical calculation, this is not very nice because of the singularity at $\mu = 2\pi$. Instead, in the molecular evolution, we use

$$\tilde{w}(\mu) = \begin{cases} 2\pi - \mu + \varepsilon & (\mu < 2\pi) \\ \varepsilon e^{-g_\alpha(\mu - 2\pi)} & (\mu \geq 2\pi), \end{cases} \tag{51}$$

with a large enough g_α (say g_α=100) and $\varepsilon = g_\alpha^{-1}$. For the Metropolis test, we use $w(\mu)$, rather than $\tilde{w}(\mu)$. Although it may look weird at first sight, it does not change the expectation values.

Projection of the $U(1)$ part is done every time a new configuration is accepted. (It is easy to see that we don't have to keep the tracelessness during the molecular evolution.)

3.5 Fourier acceleration

In the molecular evolution, we can take different time step $\Delta\tau$ for different momenta. Because low-frequency modes fluctuate more, it is better to take larger $\Delta\tau$ for them. This observation leads us to the 'Fourier acceleration' [11] – we should accelerate the fluctuation of lighter Fourier modes.

In order to implement the Fourier acceleration, we move to the momentum basis,

$$\tilde{X}_M^{ij}(k) = \frac{1}{\sqrt{N_t}} \sum_{t=1}^{N_t} X_M^{ij}(t) e^{-2\pi i k t / N_t}, \tag{52}$$

where $k = 1, 2, \cdots, N_t$. We choose the time step for the molecular evolution depending on k.

We take p as

$$p = \begin{cases} k & k \leq N_t/2 \\ k - N_t & k > N_t/2 \end{cases} \tag{53}$$

when n_t is even, and

$$p = \begin{cases} k & k \leq (N_t - 1)/2 \\ k - N_t & k > (N_t - 1)/2 \end{cases} \tag{54}$$

when N_t is odd. Because of the hermiticity, we have to take $\Delta\tau(p) = \Delta\tau(-p)$. We take $\Delta\tau(p) = \sqrt{\langle TrX_M^2(p)/N \rangle} \times \Delta\tau$ in our simulation code. (We determine this acceleration parameter from a short run without the acceleration, and then perform a long run with the acceleration.)

3.6 Random number

For the random number, we use the Mersenne twister. Subroutines can be found in 'mt19937.f90,' which is from M. Matsumoto's official webpage for the Mersenne twister at

Hiroshima University. In order to avoid unnecessary communication between MPI processes, we generate the same sequence of random numbers in each MPI process and throw away the overlapping parts. **When you modify the code, be sure to keep the streams of random numbers synchronized.**

3.7 Smearing

In order to incorporate the smearing introduced in Sec. 2.7, we prepare a subroutine **smearing_xmat** and **smearing_deriv_PF**.

The smearing of the force term is performed for **delPF_xmat** in **Calc_Force**. Note that, in **Calc_Force**, **xmat** in the pseudo-fermion part must be replaced with the smeared one. We need to change bunch of other places as well.

4 Structure of the code

4.1 Lattice-parallel and matrix-parallel versions

There are two versions,

- Lattice-parallel version. Lattice direction is divided to several sub-lattices. You should use it when you want to go to large volume with small N (say $N < 10$).
- Matrix-parallel version. In addition to the lattice direction, matrices are divided to blocks. When you want to go to large N using a lot of MPI processes, you should use it.

4.1.1 Lattice-parallelization

The lattice along the time direction is divided to sub-lattices, each containing nsite_local sites. The number of sub-lattices is the number of MPI processes, which is specified when you submit a job. The total number of sites is N_t =(number of MPI processes)×nsite_local.

4.1.2 Matrix-parallelization

Before ver.10

In order to go to larger N, we have to parallelize matrix product. We divide each matrix to (nblocks) × (nblocks) sub-matrices, as shown in Fig. 3. (nblocks is not the number of total blocks; it is the number of rows and columns.) We will denote (i, j)-th block of X by $X^{(i,j)}$. We will denote the size of the sub-matrices by nmat_block. The matrix size is nmat=(nmat_block)×(nblocks).

The lattice along the time direction is divided to sub-lattices, each containing nsite_local sites. The number of sub-lattices is denoted by nsublat. The total number of sites is

N_t =(nsublat)×nsite_local. (i,j)-block of a matrix at time t, where time t belongs isublat-th sub-lattice, is taken care by

myrank=(isublat−1)×(nblocks)^2+$(i-1)$×nblocks+$j-1$.

The total number of nodes is (nsublat)×(nblocks)^2.

We divide α to nblocks blocks as in Fig. 4. All components of α will be stored in all nodes.

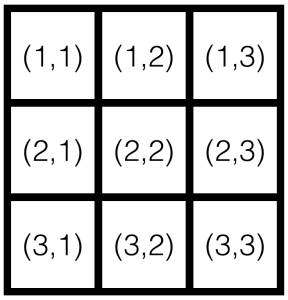


Figure 3: How the sub-blocks of a matrix is labelled. This example is for nblocks = 3.

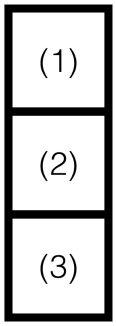


Figure 4: How the sub-vectors of α is labelled. This example is for nblocks = 3.

How to parallelize CG-solver

In the CG solver, computationally the most demanding part is the repeated multiplication of the Dirac operator to the pseudo-fermion. Note that scalars X_M, and hence the Dirac operator, does not change. Therefore, we should distribute X_M appropriately at the beginning of the solver, once and only once.

Let us put the (i,j)-block of the pseudo-fermion at (i,j)-th node[10]. Then, as we can see form Fig. 5, $X^{(k,i)}$ $(k = 1, 2, \cdots$,nblocks) should be sent to (i,j)-th node, in order to perform the multiplication from left quickly. For the same reason, in order to multiply X quickly from right, $X^{(j,k)}$ $(k = 1, 2, \cdots$,nblocks) should be sent to (i,j)-th node. (See Fig. 6.)

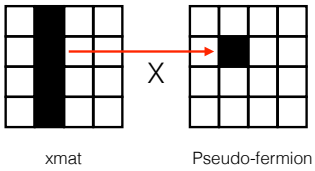


Figure 5: Multiplication of X_M to the pseudo-fermion from left. $X^{(k,i)}$ acts on $F^{(i,j)}$.

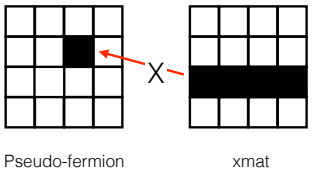


Figure 6: Multiplication of X_M to the pseudo-fermion from right. $X^{(j,k)}$ acts on $F^{(i,j)}$.

More explicitly, each multiplication of the Dirac operator on F goes as follows. To calculate $(D_t F)^{(ij)}$, we take the difference of $F^{(ij)}(t)$ and $F^{(ij)}(t+a)$, and also multiply $\alpha^{(i)}$ and $\alpha^{(j)}$. To calculate $\Gamma^M[X_M, F]$, we separate it to left and right multiplications, $\Gamma^M X_M F$ and $\Gamma^M F X_M$. At (i,j)-th node, $\Gamma^M (X_M)^{(ki)} F^{(ij)}$ and $\Gamma^M F^{(ij)} X_M^{(jk)}$ are calculated. Then we use MPI to gather the results and obtain $\Gamma^M (X_M F)^{(ij)} = \sum_k \Gamma^M X_M^{(ik)} F^{(kj)}$ and $\Gamma^M (X_M F)^{(ij)} = \sum_k \Gamma^M F^{(ik)} X_M^{(kj)}$.

Apparently, the same method can be used for the calculation of the largest and smallest eigenvalues.

[10]Of course there is also t-direction, which we do not explicitly mention here. It should be obvious!

The detail is as follows:

We want to calculate $(XF)^{(ij)} = \sum_k X^{(ik)}F^{(kj)}$ and $(FX)^{(ij)} = \sum_k F^{(ik)}X^{(kj)}$, without moving F often. (X is fixed in the CG-solver, so it is OK to move X at the first stage. On the other hand, F changes during the CG process, so it is crucial NOT to move F a lot.) So,

- Send $(X^{(i,j)})$ to (i,k)th nodes, $k = 1, 2, \cdots$. Do it for all i, $i = 1, 2, \cdots$.
- Send $(X^{(i,j)})$ to (k,j)th nodes, $k = 1, 2, \cdots$. Do it for all j, $j = 1, 2, \cdots$.
- Then the (i,j)-node has $(X^{(i,k)})$ and $(X^{(k,j)})$ for all k's. By taking, conjugate: $(X^{(k,i)}) = (X^{(i,k)})^\dagger$, $(X^{(j,k)}) = (X^{(k,j)})^\dagger$. So We can calculate $X^{(ki)}F^{(ij)}$ and $F^{(ij)}X^{(jk)}$.
- Send $(XF)^{(k,j)}$ stored in (i,j)-th node to (k,j)-th nodes, $k = 1, 2, \cdots$.
- Send $(FX)^{(i,k)}$ stored in (i,j)-th node to (i,k)-th nodes, $k = 1, 2, \cdots$.

By design, i-th and j-th blocks of α are contained in (i,j)-th node. (Actually all components are stored in all nodes.) Then both $(\alpha F)^{(ij)} = \alpha^{(i)}F^{(ij)}$ and $(F\alpha)^{(ij)} = F^{(ij)}\alpha^{(j)}$ can be calculated in the (i,j)-th node, without any communication with other nodes.

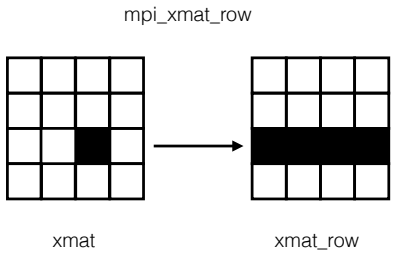


Figure 7: Subroutine mpi_xmat_row. It distributes (i,j)-block of xmat stored in (i,j)-th node to (i,k)-th nodes ($k = 1, 2, \cdots$, nblocks).

How to parallelize $X_M \times X_{M'}$

$X_M^{(i,k)}$ and $X_{M'}^{kj}$ are collected in the (i,j)-th block, and then $(X_M \times X_{M'})^{(i,j)}$ is calculated.

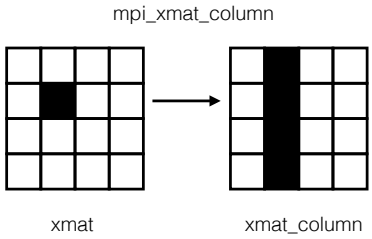


Figure 8: Subroutine mpi_xmat_column. It distributes (i, j)-block of xmat stored in (i, j)-th node to (k, j)-th nodes ($k = 1, 2, \cdots$, nblocks).

4.2 Compile

Please use **main_parallel.f90** as the main file. On my MacBook, this code can be compiled by **mpif90 main_parallel.f90 −O3**. When you compile the code, you have to specify several parameters in **size_parallel.inc**. Every time you change parameters in **size_parallel.inc**, you have to re-compile the code.

4.2.1 Parameters for compilation: lattice-parallelized version

When you compile the code, you must specify several parameters in **size_parallel.inc**. This file looks like Fig. 10. The meaning of the parameters must be obvious, but let us explain just in case:

- **nsite_local** is the number of sites at each MPI process; the total number of the sites is $N_t =$ nsite_local $\times$ (number of MPI processes), where the number of MPI processes is specified when you submit a job. The number of MPI processes must be larger than 1. (See Fig. 9.) 'nsite_local' must be ≥ 1 for nimprove $= 0$ and ≥ 2 for nimprove $= 1$, though we do not think nobody takes such ridiculously small values :).

- **nmat** is the matrix size N.

- **nremez_md** and **nremez_pf** are number of Remez coefficients in the rational approximation, for the molecular dynamics (md) and generation of the pseudo-fermion (pf). For example, when both of them are 15, the coefficients, which are specified in 'remez_md.dat' and 'remez_pf.dat,' looks like Fig. 11 and Fig. 12. Note that you need all coefficients when you compile the code.

- **ndim** is the number of scalars. It must be fixed to 9.

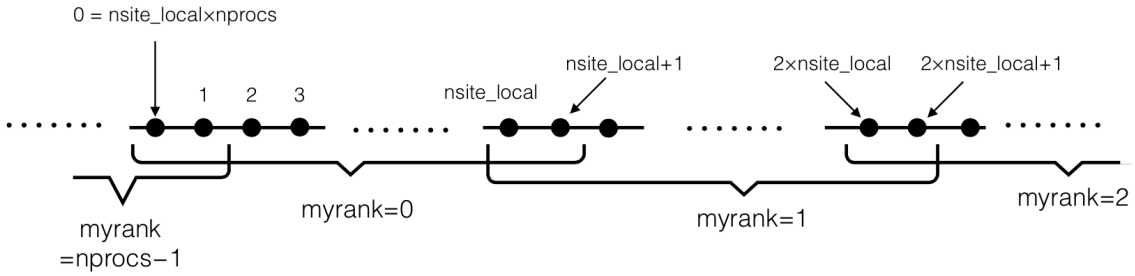


Figure 9: This figure explains how the lattice sites are distributed in the MPI processes, when 'nimprove'= 0. Total number of the lattice sites is nsite_local×nprocs, where nprocs is the number of processes which is specified when the job is submitted. MPI processes are labelled by 'myrank,' which takes values $0, 1, 2, \cdots$, nprocs. A process labeled by myrank= k manages nsite_local lattice points, $t = k\times$nsite_local$+1, \cdots, (k+1)\times$nsite_local. Because the fields at the neighboring sites interact, the fields at $t = k\times$nsite_local and $(k+1)\times$nsite_local+1 are also stored at myrank= k as margins.

- **nspin** is the size of gamma matrices. It must be fixed to 16.
- **nimprove** = 0 is the naive lattice regularization and **nimprove** = 1 is the improved regularization.
- **nmargin** is the size of the margin. For **nimprove** = 0, 1, it is **nimprove**+1. We prepared this just in case we introduce other improvement options.

Every time you change these parameters, you have to compile again.

4.2.2 Parameters for compilation: matrix-parallelized version

We also prepared 'matrix-parallelized' version, in which matrices are divided to several blocks, so that one can study very large N. Again, one has to specify several parameters in **size_parallel.inc** (Fig. 13). The meaning of the parameters are as follows:

- **nsite_local** is the number of sites at each sub-lattice, and **nsublat** is the number of sub-lattices. The total number of the sites is N_t = nsite_local × nsublat. Both nsite_local and nsublat must be larger than 1.
- Rows and columns of matrices are divided into **nblock** blocks. Size of each block is **nmat_block**, and hence N = nblock × nmat_block

Other parameters are the same as before. Note that

$$\text{number of MPI processes} = \text{nsublat} \times (\text{nblock})^2. \tag{55}$$

```
!----------------------------------------
!     Number of sites along t-direction
integer nsite_local
parameter(nsite_local=4)
!----------------------------------------
!     Size of matrices
integer nmat
parameter(nmat=4)
!----------------------------------------
!     number of remez coefficients
integer nremez_md,nremez_pf
parameter(nremez_md=15)
parameter(nremez_pf=15)
!---------------------------------------
integer ndim ! fix to 9
parameter(ndim=9)
integer nspin ! fix to 16
parameter(nspin=16)
!----------------------------------------
!   nimprove=0 -> no improvement
!   nimprove=1 -> O(a^2) improvement. Note that communication cost increases.
integer nimprove
parameter(nimprove=1)
  integer nmargin!size of the margin
parameter(nmargin=nimprove+1)
```

Figure 10: size_parallel.h for lattice-parallelized version.

```
Masanoris-MBP:fermionic_v8 masanorihanada$ cat remez_md.dat
acoeff_md(0)=6.4850604087839014e-01
acoeff_md(1)=3.6279799870602336e-12
acoeff_md(2)=2.0743547829879826e-11
acoeff_md(3)=1.2478276614820847e-10
acoeff_md(4)=7.5635579706377587e-10
acoeff_md(5)=4.5878253723142989e-09
acoeff_md(6)=2.7830157021450695e-08
acoeff_md(7)=1.6882119507840049e-07
acoeff_md(8)=1.0240909311332627e-06
acoeff_md(9)=6.2122678183206830e-06
acoeff_md(10)=3.7684495193036675e-05
acoeff_md(11)=2.2860459967530310e-04
acoeff_md(12)=1.3871227068807282e-03
acoeff_md(13)=8.4398743673200199e-03
acoeff_md(14)=5.2936835342681260e-02
acoeff_md(15)=4.6836957531388274e-01
bcoeff_md(1)=7.2190246245103699e-16
bcoeff_md(2)=1.2287943939966825e-14
bcoeff_md(3)=1.4080010963248711e-13
bcoeff_md(4)=1.5626106792756075e-12
bcoeff_md(5)=1.7292437847048868e-11
bcoeff_md(6)=1.9131519225159805e-10
bcoeff_md(7)=2.1165695948957138e-09
bcoeff_md(8)=2.3416109219624699e-08
bcoeff_md(9)=2.5905792328243406e-07
bcoeff_md(10)=2.8660219977147571e-06
bcoeff_md(11)=3.1707917451486140e-05
bcoeff_md(12)=3.5084673304121593e-04
bcoeff_md(13)=3.8881952827293570e-03
bcoeff_md(14)=4.3848134187507062e-02
bcoeff_md(15)=6.0686020612260205e-01

! result of
! ./remez.exe 1 4 15 15 1.0e-15 1.0 160
!
! approximate from 1.0e-15 to 1.0
! Q=15
!Approximation to f(x) = x^(-1/4)
```

Figure 11: remez_md.dat

```
Masanoris-MBP:fermionic_v8 masanorihanada$ cat remez_pf.dat
acoeff_pf(0)=1.2554836875790309e+00
acoeff_pf(1)=-7.8128658083565766e-17
acoeff_pf(2)=-1.0497979460686618e-15
acoeff_pf(3)=-1.3316483913742877e-14
acoeff_pf(4)=-1.6833128170609618e-13
acoeff_pf(5)=-2.1271296644548538e-12
acoeff_pf(6)=-2.6878677891115228e-11
acoeff_pf(7)=-3.3964111125975250e-10
acoeff_pf(8)=-4.2917308491166975e-09
acoeff_pf(9)=-5.4230683730199025e-08
acoeff_pf(10)=-6.8526956636776387e-07
acoeff_pf(11)=-8.6599163047192239e-06
acoeff_pf(12)=-1.0952368381019715e-04
acoeff_pf(13)=-1.3956129829123815e-03
acoeff_pf(14)=-1.9110241954162204e-02
acoeff_pf(15)=-5.3985926912539484e-01
bcoeff_pf(1)=1.1631099630541095e-14
bcoeff_pf(2)=1.4861498507042878e-13
bcoeff_pf(3)=1.4578090766973866e-12
bcoeff_pf(4)=1.3938173215882323e-11
bcoeff_pf(5)=1.3290837590187519e-10
bcoeff_pf(6)=1.2670022670997803e-09
bcoeff_pf(7)=1.2077851768080415e-08
bcoeff_pf(8)=1.1513322807536654e-07
bcoeff_pf(9)=1.0975181559366772e-06
bcoeff_pf(10)=1.0462235754261286e-05
bcoeff_pf(11)=9.9736512178956428e-05
bcoeff_pf(12)=9.5114088475318733e-04
bcoeff_pf(13)=9.1027357364440609e-03
bcoeff_pf(14)=9.0136127573996513e-02
bcoeff_pf(15)=1.2808403737231060e+00


! approximate from 1.0e-14 to 1.0
! Q=15
!Approximation to f(x) = x^(+1/8)
```

Figure 12: remez_pf.dat

4.3 Parameters specified at each run

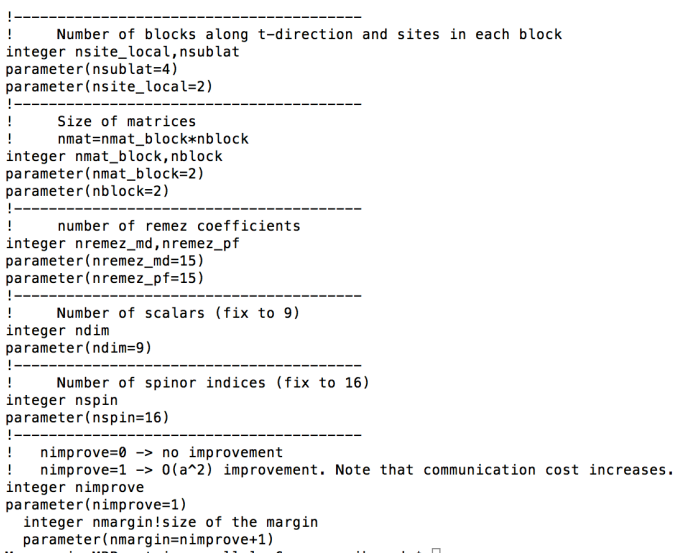
On my MacBook, it runs by **mpiexec −n 16 a.out**. Here '16' is the number of MPI processes. For the lattice-parallelized version, it can be any integer larger than 1 and smaller than the number of MPI processes available in your system. For the matrix parallelized version, it must be nsublat × $(\text{nblock})^2$.

4.3.1 Parameters specified at each run

Other parameters can be specified in[11] **input_parallel_v17.dat**, which looks like Fig. 14. The meanings are as follows:

- **input_config**: The file name of the initial configuration, which is referred when init= 0.
- **output_config**: The file name of the final configuration.
- **data_output**: Results of the measurements are written in this file. We will explain the format later.
- **intermediate_config**: The file name for the intermediate configurations.

[11]For the matrix-parallelized version, please use **input_matrix_v**.dat**, whose contents are the same.



```
!-----------------------------------------
!     Number of blocks along t-direction and sites in each block
integer nsite_local,nsublat
parameter(nsublat=4)
parameter(nsite_local=2)
!-----------------------------------------
!     Size of matrices
!     nmat=nmat_block*nblock
integer nmat_block,nblock
parameter(nmat_block=2)
parameter(nblock=2)
!-----------------------------------------
!     number of remez coefficients
integer nremez_md,nremez_pf
parameter(nremez_md=15)
parameter(nremez_pf=15)
!-----------------------------------------
!     Number of scalars (fix to 9)
integer ndim
parameter(ndim=9)
!-----------------------------------------
!     Number of spinor indices (fix to 16)
integer nspin
parameter(nspin=16)
!-----------------------------------------
!   nimprove=0 -> no improvement
!   nimprove=1 -> O(a^2) improvement. Note that communication cost increases.
integer nimprove
parameter(nimprove=1)
  integer nmargin!size of the margin
  parameter(nmargin=nimprove+1)
```

Figure 13: size_parallel.h for matrix-parralelized version.

- **acc_input**: The file name for the Fourier acceleration parameter, which is referred when iaccelerate=0. Note that you should not change the Fourier acceleration parameter if you want to combine the results of several runs to take the expectation values.

- **acc_output**: 'optimized' Fourier acceleration parameters are written in this file at the end of the run.

- **CG_log**: Numbers of CG-iterations are recorded in this file.

- **nbc**: it specifies the boundary condition for the fermion. nbc= 0 and 1 corresponds to periodic and anti-periodic (thermal) boundary conditions, respectively.

- **nbmn**: it specifies whether you simulate BFSS or BMN. nbmn= 0 and 1 corresponds to BFSS and BMN, respectively. Even with nbmn= 1, if you set flux= 0 you can study BFSS. But with nbmn= 0 the simulation runs a little bit faster.

- **init**: it specifies the initial configuration. When init= 0, the configuration in 'input_config' is used. When init= 1, the initial configuration is set to $X_M = 0$ and α_i are uniformly distributed between -1 and 1. When init= 2, the initial configuration is set to the k-coincident fuzzy sphere, $X_{1,2,3} = \mu J^{(j)}_{1,2,3} \otimes \mathbf{1}_k$ and $X_{4,5,\cdots,9} = 0$, where $k =$**nfuzzy** (see below) and the spin j is given by $N = (2j+1)k$. Again, α_i are uniformly distributed between -1 and 1.

- **iaccelerate**: when it is 0, Fourier acceleration parameters are read from acc_input. When it is 1, 'naive' ones are used.

- **isave**: when it is 0, intermediate configurations are saved every nsave trajectories.
- **temparature**: This is the inverse of the circumference of the Euclidean temporal circle β. With thermal boundary condition, this is literally temperature.
- **flux**: This is the flux parameter μ.
- **ntraj**: This is the total number of sweeps. When init= 0, the number of trajectories in the previous runs is read from the input file, and the simulation starts from there. (When 100 trajectories has been done previously, then the new run starts with the 101st trajectory.) When ntraj is smaller than the number of trajectories in the previous runs, the simulation stops immediately. When init= 1, the run starts with the first trajectory.
- **nskip**: The measurement is performed only for (nskip $\times$ n)-th trajectories, where n is integer.
- **nsave**: When isave = 0, the intermediate configurations are saved (nsave $\times$ n)-th trajectories, where n is integer.
- **ntau**: This is N_τ which is used in the molecular evolution.
- **Dtau for xmat**: This is $\Delta\tau$ for X_M.
- **Dtau for alpha**: This is $\Delta\tau$ for α. For fixed temperature, it would be better to scale $\Delta\tau_\alpha \sim 1/N$, because typical distance between neighboring α_i's scales as $1/N$, and Faddeev-Popov term forces α_i's not to come on top of each other.
- **upper_approx**: This is the upper bound of the rational approximations. The largest eigenvalue of $D = \mathcal{M}^\dagger\mathcal{M}$ must be smaller than this value.

  The Remez coefficients in 'remez_md.dat' and 'remez_pf.dat' are taken so that the rational approximation is good at $[\varepsilon, 1]$, where ε can be made small if you take 'nremez_md' and 'nremez_pf' bigger. We rescale these coeffients appropriately[12], so that the approximation is good at [upper_approx $\times\,\varepsilon$, upper_approx]. The lower limit, (upper_approx)$\times\varepsilon$, must be larger than the smallest eigenvalue of D.
- **max_err**: This specifies the stopping condition for CG solver. We solve $(D+b_k)\chi_k = F$ iteratively. The iteration stops when the relative error $|F-(D+b_k)\chi_k|/|F|$ becomes smaller than max_err.
- **max_iteration**: This sets the maximum number of the iteration in the CG solver. We introduced it in order to avoid the endless loop.

[12]Suppose the rational approximation $x^p \simeq a_0 + \sum_k \frac{a_k}{x+b_k}$ is valid at $\varepsilon \le x \le 1$. Then, for $\tilde{x} \equiv cx$, the approximation $\tilde{x}^p \simeq c^p a_0 + \sum_k \frac{c^{p+1}a_k}{\tilde{x}+cb_k}$ is valid at $c\varepsilon \le \tilde{x} \le c$.

- **g_alpha**: Needless to say, this is g_α.
- **g_R**: g_R
- **RCUT**: This means R, which is used to control the flat direction.
- **neig_max**: The number of multiplications of D in order to estimate the largest eigenvalue. When neig_max= 0, the largest eigenvalue is not calculated.
- **neig_min**: The number of multiplications of D in order to estimate the smallest eigenvalue. When neig_min= 0, the smallest eigenvalue is not calculated.
- **nfuzzy**: number of fuzzy spheres, used only when **init**= 2.
- **mersenne_seed**: seed of the Mersenne twister, used when **init**$\neq$ 0.
- **imetropolis**: when **imetropolis**= 1, the Metropolis test is skipped. (Still, when the CG-solver fails to converge or the constraint $|\alpha_i - \alpha_j| < 2\pi$ is violated, new configuration is rejected.) This option can be used to reach the thermalization quickly.

```
'conN4S16T20_apbc.dat'          !input_config
'conN4S16T20_apbc_2.dat'        !output_config
'outN4S16T20_apbc_2.txt'        !data_output
'iconN4S16T20_apbc_2.dat'       !intermediate_config
'acc_N4S16T20_apbc.dat'         !acc_input
'acc_N4S16T20_apbc_2.dat'       !acc_output
'CG_N4S16T20_apbc_2.dat'       !CG_log
1       !nbc; 0 -> pbc, 1 -> apbc
0       !nbmn; 0 -> BFSS, 1 -> BMN
0       !init; 0 -> old config, 1 -> new config, 2-> fuzzy sphere
0       !iaccelerate; 0 -> read from acc_input, 1-> naive
1       !isave; 0 -> save intermediate config, 1-> do not save
2.0d0   !temperature
1.0d0   !flux parameter mu; used only when nbmn=1
200     !ntraj(total number of sweeps)
1       !nskip
2       !nsave
5       !Ntau
0.03d0  !Dtau for xmat
0.03d0  !Dtau for alpha
1d3     !upper_approx; the largest eigenvalue of (M^¥dagger M)must be smaller than this.
1d-10   !max_err; stopping condition for CG solver
500     !max_iteration; maximum number of CG-iteration
1000d0  !g_alpha; coefficient for constraint term of alpha
100d0   !g_R; coefficient for constraint term of R
100.0d0 !RCUT; cutoff for TrX^2
10      !neig_max; neig_max>0 -> calculate the largest eig of (D^dag*D).
10      !neig_min; neig_min>0 -> calculate the smallest eig of (D^dag*D).
1       !nfuzzy; number of fuzzy spheres. Used only when init=2. mod(nmat,nfuzzy) must be zero.
4357    !mersenne_seed; seed for mersenne twistor, when init is not zero.
1       !imetropolis; 1 -> no Metropolis test (For thermalization)
```

Figure 14: input_parallel_v17.dat

4.4 output of the measurement results

The output format is specified in a subroutine **output_header**. At the header lines, boundary condition (pbc or apbc), BFSS or BMN, and various simulation parameters are explained. The meanings of the numbers are explained at the end of the header; the first column is itraj, the sixth column is the interenal energy, etc.

An example is shown in Fig. 15.

- Here, the final configuration of the previous run has been recorded in 'conN4S16T20_apbc.dat,' and we used it as the initial configuration for this run. 100 trajectories have been done so far, so this run starts with itaj = 101.
- $\frac{1}{N}\int dtTrX^2$ is fluctuating between 3 and 4, while 'RCUT' is taken to be 100. This means the flat direction is well under control without adding additional potential term.
- It takes at most 231 iterations for the CG solver to converge; hence our choice of the maximum allowed CG iterations, 500, can reasonably tame a possible non-convergent event coming from numerical instability, without altering 'healthy' events. (These CG-iterations in the output file is the last one in each RHMC-evolution. If you want to check all CG-iterations during the simulation, you should see 'CG_N4S16T20_apbc_2.dat.')
- In our previous experience, simulation is the most effective (i.e. the number of independent configurations, which do not suffer from the auto-correlation, is maximized) when $N_\tau\Delta\tau$ is around 0.1 – 0.3 and the acceptance rate is 0.7 – 0.9. We chose N_τ and $\Delta\tau$ following this rule. Other values would be favored in at very low temperature and/or very large N. (At first ten trajectories in this run the acceptance was 1.0, but it will eventually comes down.)
- We used a set of Fourier acceleration parameters recorded in 'acc_N4S16T20_apbc.dat.' If you want to combine the data from several runs, you have to make sure all parameters, including the Fourier acceleration, are the same.
- The upper limit of the Remez approximation (rational approximation for $D^\dagger D^{1/8}$ and $D^\dagger D^{-1/4}$) are taken to be 1000. With the coefficients shown in Fig. 11 and Fig. 12, the lower bounds are this value times 10^{-15} for $D^\dagger D^{-1/4}$ and 10^{-14} for $D^\dagger D^{1/8}$. The largest eigenvalue (10th column) and smallest eigenvalue (11th column) are in this range, so the approximation is valid.

4.5 The main routine

In the main file **main_parallel.f90**, after the input parameters are read, several subroutines follows:

```
#apbc
#BFSS matrix model
#improved lattice action
#size of the gauge group: nmat=           4
#size of the lattice: Nt=          16
#number of MPI processes in this run:           4
#Temperature=   2.0000000000000000
#g_alpha=   1000.0000000000000
#g_R=   100.00000000000000
#RCUT=   100.00000000000000
#(TrX^2 is constrained so that TrX^2 < RCUT)
#ntau=           5
#dtau for xmat=   2.9999999999999999E-002
#dtau for alpha=   2.9999999999999999E-002
#CG stopping condition: relative error=   1.0000000000000000E-010
#CG stopping condition: maximum CG iteration=         500
#upper limit of remez approximation (Largest eigenvalue of D^dag*D must be smaller than this):   1000.0000000000000
#init=           0
#(init; 0 -> old config, 1 -> new config, 2-> fuzzy sphere)
#imetropolis=           1
#(imetropolis=1 -> no Metropolis test,just for thermalization)
#input configuration:conN4S16T20_apbc.dat
#output configuration:conN4S16T20_apbc_2.dat
#iaccelerate=           0
#(iaccelerate; 0 -> read from acc_input, 1-> naive)
#Fourier acceleration file used for this run:acc_N4S16T20_apbc.dat
#Fourier acceleration file obtained from this run:acc_N4S16T20_apbc_2.dat
#log of the CG solver is saved in:CG_N4S16T20_apbc_2.dat
# iteration for evaluating largest eig=          10
# iteration for evaluating smallest eig=          10
# traj,ham_fin-ham_init, number of constraint violation,number of CG-err,CG-iteration,energy, |Pol.|, trX^2, tr[X,Y]^2, largest eig,smallest eig, acceptance
#-------------------------------------------------
    101    -0.188664912    0       0      218      10.7406615       0.9631578       3.5937030       17.9917893      15.9504747      0.0258470       1.0000000
    102     0.963997165    0       0      196      10.0456884       0.9833995       4.0915922       24.5213763      15.9524154      0.0310862       1.0000000
    103     0.584320256    0       0      199      28.1608240       0.9752040       4.0375135       22.6450374      15.9909613      0.0292193       1.0000000
    104    -0.444909097    0       0      212      -0.6051657       0.9696316       4.4863530       23.6722578      16.0061264      0.0261720       1.0000000
    105     0.292091407    0       0      212      10.8764881       0.9705585       4.4442110       22.4509685      16.0643626      0.0275270       1.0000000
    106     0.339136725    0       0      182      27.5246570       0.9864037       4.0356479       23.4859203      15.9438378      0.0304372       1.0000000
    107    -0.028348272    0       0      195      25.7029480       0.9820977       3.6208911       20.6309934      15.9458274      0.0299845       1.0000000
    108    -0.262220791    0       0      231      12.0376908       0.9567812       3.3092707       15.5639008      15.9431199      0.0252013       1.0000000
    109     0.642142267    0       0      213      11.6759211       0.9695114       4.3984148       33.6276368      16.0200352      0.0273898       1.0000000
    110     0.656291402    0       0      184      22.6604072       0.9816341       3.6204277       19.4109665      15.9506566      0.0298622       1.0000000
```

Figure 15: Output of the measurement results.

- **initial_configuration** sets the initial configuration. If you want to study a specific vacuum which is not prepared in the original version of this code, you should add the corresponding initial configuration here.
- In **output_header** parameters for the run is written in the output file.
- **RHMC_evolution** takes care of the molecular evolution, including the Metropolis test.
- **hermitian_projection** prevents xmat to deviates from Hermitian due to accumulated numerical errors.
- Measurement subroutines are called in **measurements**.
- **Fourier_acceleration_optimize** calculates the optimized acceleration parameter, so that we can use it in the future runs.
- In **Save_Final_Config** the final configuration and random number are recorded.
- In **Save_Acceleration_Parameters** the optimized Fourier acceleration parameters are recorded.

Various subroutines are called inside these subroutines.

4.6 Subroutines

4.6.1 For configuration generation

- **alpha_constraint**.

  It checks whether the constraint $\max(\tilde{\alpha}_i) - \min(\tilde{\alpha}_i) < 2\pi$ is satisfied.

- **Calc_action**.

  It calculates the action at each rank. The gauge fixing term is calculated at myrank = 0.

- **Calc_Force**.

  It calculates the derivative of the Hamiltonian (force times (-1)) with respect to the dynamical variables, at each MPI process. It is called in"Molecular_Dynamics_gauge_fixed_local."

- **Calc_Ham**.

  It calculates the Hamiltonian at each rank. The gauge fixing term and $P_\alpha^2/2$ are calculated at myrank = 0. It calls "Calc_action_gauge_fixed_local".

- **Derivative_Dirac_alpha**.

  It calculates $\frac{\partial \mathcal{M}}{\partial \alpha} v$, where v is a vector.

- **Derivative_Dirac_X**.

  It calculates $\frac{\partial \mathcal{M}}{\partial X} v$, where v is a vector.

- **Fourier_acceleration_naive**.

  It is used for setting the Fourier acceleration parameter.

- **Fourier_acceleration_optimize**.

  It is used for setting the Fourier acceleration parameter.

- **Fourier_transform_xmat, Fourier_transform_P_xmat**.

  They are used for the Fourier transformations, $X(t) \to \tilde{X}(k)$ and $P_X(t) \to \tilde{P}_X(k)$.

- **Generate_P_alpha**.

  It generates P_α by using "BoxMuller".

- **Generate_P_xmat**.

  It generates P_X by using "BoxMuller". It is called at myrank = 0 and then P_X is distributed to all processes.

- **Generate_pseudo_fermion_SUN**.

  It generates the pseudo-fermion F which satisfies the traceless condition. (SUN means $SU(N)$.)

- **Molecular_Dynamics**.

  It performs the molecular evolution, by using the leap frog method.

- **Multiply_Dirac**.

  It multiplies the Dirac operator $\mathcal{M}$ to a vector. It is called in Calc_DELH_gauge_fixed_local and solver_biCGm.

- **Multiply_Dirac_dagger**.

  It multiplies the hermitian conjugate of the Dirac operator, $\mathcal{M}^\dagger$, to a vector. It is called in Calc_Force and solver_biCGm.

- **Multiply_Dirac_to_chi**.

  It multiplies the Dirac operator $\mathcal{M}$ to χ_k. It is called in Calc_Force.

- **solver_biCGm**.

  This is a multi-mass CG solver, which calculates χ_k from F. It uses Multiply_Dirac_local and Multiply_Dirac_dagger_local.

- **subtract_U1**.

  Subtract the U(1) part of α and $\int dtX$.

4.6.2 For measurements

- **Calc_Com2**.

  It calculates $-\frac{1}{N\beta}\sum_{M\neq M'}\int dt[X_M, X_{M'}]^2$

- **Calc_energy**.

  It calculates the energy $E = -\frac{\partial}{\partial\beta}\log Z$ of BFSS/BMN matrix model.

  Previously, for BFSS, we used $E = \frac{3}{2\beta}\{9(N^2N_t - 1) - 2\langle S_b\rangle\}$.

  After matrix v10.4, we use the following expression following Ref. [17]:

  By using the virial theorem $\langle K\rangle = \frac{1}{2}\left\langle X\frac{\partial V}{\partial X}\right\rangle$, where K and V are kinetic and potential parts of the Hamiltonian (note that the kinetic term of the Hamiltonian consists of bosons only; fermion term is absent, different from the Lagrangian.), we can easily obtain

$$\frac{E}{N^2} = \left\langle \frac{1}{N\beta}\int_0^\beta dt\ \mathrm{Tr}\left\{ -\frac{3}{4}[X_M, X_N]^2 + \mu^2\sum_{i=1}^{3}X_i^2 + \frac{\mu^2}{4}\sum_{a=4}^{9}X_a^2 + \frac{5i\mu}{2}\sum_{i,j,k=1}^{3}\epsilon^{ijk}X_iX_jX_k \right.\right.$$

$$-\frac{3}{2}\bar{\psi}\gamma^M[X_M,\psi]+\frac{3i\mu}{4}\bar{\psi}\gamma^{123}\psi\Bigg\}\Bigg\rangle. \tag{56}$$

The fermionic term can be rewritten as

$$\begin{aligned}
&\left\langle \frac{1}{N\beta}\int_0^\beta dt\ \mathrm{Tr}\left\{-\frac{3}{2}\bar{\psi}\gamma^M[X_M,\psi]+\frac{3i\mu}{4}\bar{\psi}\gamma^{123}\psi\right\}\right\rangle \\
&= -\left\langle \frac{1}{N\beta}\mathrm{Tr}\left(\mathcal{M}^{-1}\left\{-\frac{3}{2}\gamma^M[X_M,\ \cdot\ ]+\frac{3i\mu}{4}\gamma^{123}\right\}\right)\right\rangle \\
&\equiv \left\langle \frac{1}{N\beta}\mathrm{Tr}\left(\mathcal{M}^{-1}K\right)\right\rangle
\end{aligned} \tag{57}$$

where $S_f=\frac{1}{2}\mathcal{M}_{A\alpha r;B\beta s}\tilde{\psi}^A_{\alpha r}\tilde{\psi}^B_{\beta s}$ and $K=\frac{3}{2}\gamma^M[X_M,\ \cdot\ ]-\frac{3i\mu}{4}\gamma^{123}$. It can be rewritten further as

$$\left\langle \frac{1}{N\beta}\mathrm{Tr}\left(\mathcal{M}^{-1}K\right)\right\rangle=\left\langle \frac{1}{N\beta}\Phi^\dagger\mathcal{M}^{-1}K\Phi\right\rangle \tag{58}$$

where $\langle\ \cdot\ \rangle$ on the right hand side represents the average over complex Gaussian random matrices with the weight $e^{-\Phi^\dagger\Phi}$. We will use one realization of Φ at each measurement of the energy.

We proceed as follows:

(i-1) generate Φ; (i-2) multiply K; (i-3) multiply $\mathcal{M}^\dagger$. Then we obtain $\mathcal{M}^\dagger K\Phi$.

(ii) solve $\mathcal{M}^\dagger\mathcal{M}\chi=\Phi$ to obtain $\chi=\mathcal{M}^{-1}(\mathcal{M}^\dagger)^{-1}\Phi$ by using **solver_biCGm**.

(iii) take the inner product of $\mathcal{M}^\dagger K\Phi$ and χ, to obtain $\chi^\dagger\mathcal{M}^\dagger K\Phi=\Phi^\dagger\mathcal{M}^{-1}K\Phi$.

Note that '$\mathcal{M}$' defined in the code has different normalization; it is actually $-\frac{ia}{2}\mathcal{M}$.

- **Calc_Polyakov**.

  It calculates the Polakov loop, $|P|=|\sum_{j=1}^N e^{i\alpha_j}|$.

- **Calc_TrX2**.

  It calculates $\frac{1}{N\beta}\sum_{M\neq M'}\int dtX_M^2$.

4.6.3 MPI communications

- **Adjust_margin_xmat**.

  It adjusts the margin of xmat.

- **Adjust_margin_and_bc_pf**.

  It adjusts the margin and the boundary condition of the pseudo-fermion.

- **Adjust_margin_and_bc_Chi**.

  It adjusts the margin and the boundary condition of χ.

4.6.4 Random number generator

- **BoxMuller**.

  It generates Gaussian random number by using Box-Muller algorithm.

- **mtgetu**, **mtsaveu**.

  Read and save the information about the sequence of the random number. These routines are from M. Matsumoto's official webpage for the Mersenne twister at Hiroshima University.

- **sgrnd**.

  Set the seed for the Mersenne twister. This routines are from M. Matsumoto's official webpage for the Mersenne twister at Hiroshima University.

4.6.5 Others

- **MakeGamma**.

  It constructs the Gamma matrices.

- **Largest_eigenvalue**.

  When neig_max > 0, it calculates the largest eigenvalue of $D = \mathcal{M}^\dagger \mathcal{M}$. The eigenvalue is calculated approximately, by $|D^{\text{neig\_max}} v|/|D^{\text{neig\_max}-1} v|$, where v is a random vector.

- **Smallest_eigenvalue**.

  When neig_min > 0, it calculates the largest eigenvalue of $D = \mathcal{M}^\dagger \mathcal{M}$. The eigenvalue is calculated approximately, by $\left(|(D^{-1})^{\text{neig\_min}} v|/|(D^{-1})^{\text{neig\_min}-1} v|\right)^{-1}$, where v is a random vector.

4.6.6 Subroutines specific to the matrix-parallelized version

- **matrix_format_change**. Format change from block-divided matrices to a big matrix, and vice versa.

- **mpi_xmat_column**. In collects the information of (k, j)-blocks of xmat ($k = 1, 2, \cdots$, nblock) in the (i, j)-block.

- **mpi_xmat_row**. In collects the information of (i, k)-blocks of xmat ($k = 1, 2, \cdots$, nblock) in the (i, j)-block.

- **who_am_i**. It returns (isublat, iblock, jblock) such that myrank $=$ (isublat $-$ 1) $\cdot$ (nblock)^2 + (iblock $-$ 1) $\cdot$ nblock + jblock $-$ 1. Literally, it teaches the processes who they are.

5 Machine-dependent tuning for speedup

Usually, the most time-consuming part, both in terms of the calculation and the communication, is multiplication of the Dirac operator and its dagger in the CG-solver (**Multiply_Dirac** and **Multiply_Dirac_dagger**). In particular, the multiplication of scalar matrix X_M and pseudofermion is the bottle neck. If you can use a package like ScaLAPACK, you might be able to speed up the matrix multiplication by using it.

Acknowledgements

I would like to thank many people:

– A. Tsuchiya, who implemented OpenMP to this code. He optimized it on K-supercomputer.

– I. Kanamori, who carefully looked at the code, which was originally not very much readable, and gave me various suggestions for increasing the readability.

– Y. Ito, T. Kaneko, J. Nishimura and A. Tsuchiya for collaboration in the algorithm building and cross-checking at earlier stage of the code development.

– S. Shenker and L. Susskind for great hospitality, and their paper [1], without which not just this code but entire field of computational quantum gravity would not even exist.

– E. Berkowitz, G. Ishiki, S. Matsuura E. Rinaldi and S. Shimasaki. They tested the code on various machines, thanks to which I could remove a few environment-dependent bugs.

– P. Vranas for suggesting a few technical improvements.

– A. Nakamura, who gave me valuable advices based on the experience from his open-source code, *Lattice Tool Kit*.

– S. Catterall and D. Schaich, for advices based on the experience from their open-source code for lattice 4d $\mathcal{N} = 4$ SYM.

– D. Kawai for pointing out several typos in equations in this manual.

– H. Lin, J. Penedones and X. Yin; I decided to write this code partly because the old one was not capable of the projects they suggested.

– Last but not least, I should better thank my parents, who somehow came up with a first name starting with M.

A Multi-mass BiCG method

A.1 (Single-mass) BiCG method

Let us first introduce the ordinary (or 'single-mass') biconjugate gradient method. The notation is that of [12]. We want to solve the linear equation

$$A\vec{x} = \vec{b}. \tag{59}$$

We construct a sequence of approximate solutions $\vec{x}_1, \vec{x}_2, \cdots$ which (almost always) converges to the solution.

We start with an initial trial solution $\vec{x}_1$, which is arbitrary. From this we define $\vec{r}_1, \vec{\tilde{r}}_1, \vec{p}_1, \vec{\tilde{p}}_1$ as

$$\vec{r}_1 = \vec{\tilde{r}}_1 = \vec{p}_1 = \vec{\tilde{p}}_1 = \vec{b} - A\vec{x}_1. \tag{60}$$

Then, we construct $\vec{x}_k$ as follows:

- $\alpha_k = \frac{\vec{\tilde{r}}_k \cdot \vec{r}_k}{\vec{\tilde{p}}_k \cdot A\vec{p}_k}$.
- $\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{p}_k$.
- $\vec{r}_{k+1} = \vec{r}_k - \alpha_k A\vec{p}_k$,
  $\vec{\tilde{r}}_{k+1} = \vec{\tilde{r}}_k - \alpha_k A^T \vec{\tilde{p}}_k$.
- $\beta_k = \frac{\vec{\tilde{r}}_{k+1} \cdot \vec{r}_{k+1}}{\vec{\tilde{r}}_k \cdot \vec{r}_k}$.
- $\vec{p}_{k+1} = \vec{r}_{k+1} + \beta_k \vec{p}_k$,
  $\vec{\tilde{p}}_{k+1} = \vec{\tilde{r}}_{k+1} + \beta_k \vec{\tilde{p}}_k$.

Note that $\vec{r}_k = \vec{b} - A\vec{x}_k$. Usually, the norm of $\vec{r}_k$ converges to zero, or equivalently, $\vec{x}_k$ converges to the solution.

A.2 Multi-mass BiCG method

Let A_σ be 'shifted' version of A:

$$A_\sigma = A + \sigma \cdot \mathbf{1}. \tag{61}$$

If A were the Laplacian, σ would be interpreted as a mass term. The multi-mass BiCG solver [9] enables us to solve $A_\sigma \vec{x} = \vec{b}$ for many different values of σ simultaneously, with only negligibly small additional cost.

The key idea is that, from the iterative series for A,

$$\begin{aligned} \vec{r}_{k+1} &= \vec{r}_k - \alpha_k A\vec{p}_k, \\ \vec{p}_{k+1} &= \vec{r}_{k+1} + \beta_k \vec{p}_k, \end{aligned} \tag{62}$$

it is possible to construct a similar series for the shifted operator,

$$\begin{aligned} \vec{r}^{\,\sigma}_{k+1} &= \vec{r}^{\,\sigma}_k - \alpha^\sigma_k A_\sigma \vec{p}^{\,\sigma}_k, \\ \vec{p}^{\,\sigma}_{k+1} &= \vec{r}^{\,\sigma}_{k+1} + \beta^\sigma_k \vec{p}^{\,\sigma}_k, \end{aligned} \tag{63}$$

where

$$\vec{r}^{\,\sigma}_k = \zeta^\sigma_k \vec{r}_k. \tag{64}$$

Indeed, (63) can be satisfied by taking[13]

$$\begin{aligned}
\alpha_k^\sigma &= \alpha_k \cdot \frac{\zeta_{k+1}^\sigma}{\zeta_k^\sigma}, \\
\beta_k^\sigma &= \beta_k \cdot \left(\frac{\zeta_{k+1}^\sigma}{\zeta_k^\sigma}\right)^2, \\
\zeta_{k+1}^\sigma &= \frac{\zeta_k^\sigma \zeta_{k-1}^\sigma \alpha_{k-1}}{\alpha_{k-1}\zeta_{k-1}^\sigma(1+\alpha_k\sigma) + \alpha_k\beta_{k-1}(\zeta_{k-1}^\sigma - \zeta_k^\sigma)}.
\end{aligned} \tag{66}$$

Therefore, we can generalize the usual BiCG solver in the following manner:

- $\alpha_k = \frac{\vec{\bar{r}}_k \cdot \vec{r}_k}{\vec{\bar{p}}_k \cdot A\vec{p}_k}$.
- Calculate ζ_{k+1}^σ and α_k^σ using (66).
- $\vec{x}_{k+1}^\sigma = \vec{x}_k^\sigma + \alpha_k^\sigma \vec{p}_k^\sigma$.
- $\vec{r}_{k+1} = \vec{r}_k - \alpha_k A\vec{p}_k$,
  $\vec{\bar{r}}_{k+1} = \vec{\bar{r}}_k - \alpha_k A^T \vec{\bar{p}}_k$.
- $\beta_k = \frac{\vec{\bar{r}}_{k+1} \cdot \vec{r}_{k+1}}{\vec{\bar{r}}_k \cdot \vec{r}_k}$.
- Calculate β_k^σ using (66).
- $\vec{p}_{k+1} = \vec{r}_{k+1} + \beta_k \vec{p}_k$,
  $\vec{\bar{p}}_{k+1} = \vec{\bar{r}}_{k+1} + \beta_k \vec{\bar{p}}_k$,
  $\vec{p}_{k+1}^\sigma = \vec{r}_{k+1}^\sigma + \beta_k^\sigma \vec{p}_k^\sigma$.

Then, $\vec{x}_k^\sigma$ is an approximate solution with the residual vector $\vec{r}_k^\sigma$,

$$\vec{r}_k^\sigma = \vec{b} - A_\sigma \vec{x}_k^\sigma. \tag{67}$$

Note that we cannot start with an arbitrary initial condition, because (63) is not satisfied then. We choose the following special initial condition in order to satisfy (63):

$$\begin{aligned}
&\vec{x}_1 = \vec{x}_1^\sigma = \vec{p}_0 = \vec{p}_0^\sigma = \vec{0}, \\
&\vec{r}_1 = \vec{r}_1^\sigma = \vec{r}_0 = \vec{r}_0^\sigma = \vec{p}_1 = \vec{p}_1^\sigma = \vec{b}, \\
&\zeta_0^\sigma = \zeta_1^\sigma = \alpha_0 = \alpha_0^\sigma = \beta_0 = \beta_0^\sigma = 1.
\end{aligned} \tag{68}$$

As long as we stick to this initial condition, it is hard to implement any preconditioning. If you know any preconditioning which can be used with multi-mass CG solver, please let me know.

[13] From (62) we have

$$\vec{r}_{k+1} = \left(1 + \frac{\alpha_k\beta_{k-1}}{\alpha_{k-1}}\right)\vec{r}_k - \alpha_k A\vec{r}_k - \frac{\alpha_k\beta_{k-1}}{\alpha_{k-1}}\vec{r}_{k-1}. \tag{65}$$

Comparing the coefficients with those in a similar equation obtained from (63), we obtain (66).

References

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

[2] B. de Wit, J. Hoppe and H. Nicolai, "On the Quantum Mechanics of Supermembranes," Nucl. Phys. B **305**, 545 (1988).

[3] D. E. Berenstein, J. M. Maldacena and H. S. Nastase, "Strings in flat space and pp waves from N=4 superYang-Mills," JHEP **0204**, 013 (2002) [hep-th/0202021].

[4] N. Itzhaki, J. M. Maldacena, J. Sonnenschein and S. Yankielowicz, "Supergravity and the large N limit of theories with sixteen supercharges," Phys. Rev. D **58**, 046004 (1998) [hep-th/9802042].

[5] J. M. Maldacena, M. M. Sheikh-Jabbari and M. Van Raamsdonk, "Transverse five-branes in matrix theory," JHEP **0301**, 038 (2003) [hep-th/0211139].

[6] T. Ishii, G. Ishiki, S. Shimasaki and A. Tsuchiya, Phys. Rev. D **78**, 106001 (2008) [arXiv:0807.2352 [hep-th]].

[7] M. Hanada, T. Kaneko, J. Nishimura and A. Tsuchiya, to appear in hep-th.

[8] M.A. Clark and A.D. Kennedy,
`http://www.ph.ed.ac.uk/~mike/remez,2005`.

[9] B. Jegerlehner, "Krylov space solvers for shifted linear systems," arXiv:hep-lat/9612014.

[10] M. A. Clark and A. D. Kennedy, "The RHMC algorithm for 2 flavors of dynamical staggered fermions," Nucl. Phys. Proc. Suppl. **129**, 850 (2004) [arXiv:hep-lat/0309084].

[11] S. Catterall and S. Karamov, "Testing a Fourier accelerated hybrid Monte Carlo algorithm," Phys. Lett. B **528**, 301 (2002).

[12] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, "Numerical Recipes in FORTRAN, " Cambridge University Press.

[13] K. N. Anagnostopoulos, M. Hanada, J. Nishimura and S. Takeuchi, "Monte Carlo studies of supersymmetric matrix quantum mechanics with sixteen supercharges at finite temperature," Phys. Rev. Lett. **100**, 021601 (2008) [arXiv:0707.4454 [hep-th]].

[14] M. Hanada, Y. Hyakutake, J. Nishimura and S. Takeuchi, Phys. Rev. Lett. **102**, 191602 (2009) [arXiv:0811.3102 [hep-th]].

[15] M. Hanada, J. Nishimura and S. Takeuchi, "Non-lattice simulation for supersymmetric gauge theories in one dimension," Phys. Rev. Lett. **99**, 161602 (2007) [arXiv:0706.1647 [hep-lat]].

[16] S. Catterall and T. Wiseman, "Black hole thermodynamics from simulations of lattice Yang-Mills theory," Phys. Rev. D **78**, 041502 (2008) [arXiv:0803.4273 [hep-th]].

V. G. Filev and D. O'Connor, "The BFSS model on the lattice," arXiv:1506.01366 [hep-th].

[17] Y. Asano, V. G. Filev, S. Kov?ik and D. O'Connor, "The non-perturbative phase diagram of the BMN matrix model," JHEP **1807**, 152 (2018) [arXiv:1805.05314 [hep-th]].