September 6, 2024

Variational Monte Carlo code for BFSS matrix model

Masanori Hanada

School of Mathematical Sciences, Queen Mary University of London
Mile End Road, London, E1 4NS, United Kingdom

Abstract

I explain the variational Monte Carlo simulation code for the bosonic matrix model. Additionally, I discuss how supersymmetry can be incorporated, although the code for the supersymmetric model will be provided separately.

For detailed information on the bosonic matrix model and the variational Monte Carlo method, please refer to Ref. [1].

There may be typos; please let me know if you find any.

Contents

1 BNAF for bosonic matrix model

1.1 Obtaining ground-state wave function

Let us split the wave function into an absolute value and a phase as

$$\Psi(X) = \psi(X) \cdot e^{\mathrm{i}\varphi(X)}, \qquad \psi(X) = |\Psi(X)| \geq 0\,. \tag{1}$$

We introduce neural networks for $\psi(X)$ and $\varphi(X)$ separately.

To obtain the ground state, we minimize the energy $E[\Psi]$ defined by

$$\begin{aligned} E[\Psi] &= \int \mathrm{d}X \Psi^*[\hat{H}\Psi] \\ &= \int \mathrm{d}X\ \psi^2 \left(\frac{1}{2}(\partial_X \log\psi)^2 + \frac{1}{2}(\partial_X\varphi)^2 + V(X) \right). \end{aligned} \tag{2}$$

Obviously, $\varphi =$ constant is required for the ground state. Therefore, we will ignore the phase factor φ below.[1] (We will include φ when we study excited states.)

Because $|\psi_\theta(X)|^2$ gives the probability distribution, we want to take it in a form convenient for the Monte Carlo integral in the VMC algorithm, i.e. a form that it is easy to sample from. We use a neural network for this purpose, adopting the Block Neural Autoregressive Flow (BNAF) architecture [2].

A (neural) normalizing flow works by taking a simple distribution $p_0(\vec{z})$ for $\vec{z} \in \mathbb{R}^d$ with $d = D(N^2-1)$ ($D =$ `ndim` is the number of matrices) in our case (say a multivariate Gaussian distribution) and defining a one-to-one map (bijective transformation)

$$\vec{x} = \vec{f}_\theta(\vec{z}) \tag{3}$$

[1] Regarding the singlet condition: The ground state is SU(N)-invariant. We can easily show that, if $\psi e^{\mathrm{i}\varphi}$ is SU(N)-invariant, then both ψ and φ are SU(N) invariant.

by using a neural network, where $x_i = X_1^i$, $x_{N^2-1+i} = X_2^i$ for $i = 1, \cdots, N^2-1$, ..., $x_{(D-1)(N^2-1)+i} = X_D^i$ for $i = 1, \cdots, N^2-1$. Below, let us use a notation $\vec{x}_\theta$ to emphasize that $\vec{x} = \vec{f}_\theta(\vec{z})$ depends on the parameters θ.

By using the Jacobian matrix

$$J_{\theta,ij} \equiv \frac{\partial x_{\theta,j}}{\partial z_i}\,, \tag{4}$$

we can write

$$p_0(\vec{z})\mathrm{d}\vec{z} = p_0(\vec{z})\left|\det J_\theta\right|^{-1}\mathrm{d}\vec{x}\,, \tag{5}$$

we can relate the map $\vec{f}_\theta$ and the wave function ψ_θ as

$$|\psi_\theta(\vec{x})|^2 = \frac{p_0(\vec{z})}{\det J_\theta}\,. \tag{6}$$

The energy is

$$\begin{aligned} E[\psi_\theta] &= \int \mathrm{d}\vec{z} p_0(\vec{z}) \left(\frac{1}{2}(\partial_{x_\theta}\log\psi(\vec{x}_\theta(\vec{z})))^2 + V(\vec{x}_\theta(\vec{z}))\right) \\ &= \mathbb{E}_{\vec{z}\sim p_0(\vec{z})}\left[\frac{1}{2}(\partial_{x_\theta}\log\psi(\vec{x}_\theta(\vec{z})))^2 + V(\vec{x}_\theta(\vec{z}))\right]\,. \end{aligned} \tag{7}$$

The derivative ∂_{x_θ} is written in terms of Jacobian as

$$\frac{\partial}{\partial x_{\theta,i}} = J^{-1}_{\theta,ij}\frac{\partial}{\partial z_j}\,. \tag{8}$$

Practically, we can simply generate $\vec{z}$ with a probability $p_0(\vec{z})$ and take the ensemble average of $\epsilon_\theta(\vec{z}) \equiv \frac{1}{2}(\partial_{x_\theta}\log\psi(\vec{x}_\theta(\vec{z})))^2 + V(\vec{x}_\theta(\vec{z}))$ and $\nabla_\theta\epsilon_\theta(\vec{z})$ to obtain $E_\theta \equiv E[\psi_\theta]$ and $\nabla_\theta E_\theta$, respectively.

SU(N) charge

$$\langle\psi_\theta|\,\hat{G}^2\,|\psi_\theta\rangle = \int \mathrm{d}\vec{x}(\hat{G}\psi_\theta)^2 = \int \mathrm{d}\vec{x}\psi_\theta^2(\hat{G}\log\psi_\theta)^2 = \mathbb{E}_{\vec{z}\sim p_0(\vec{z})}\left[(\hat{G}\log\psi_\theta)^2\right] \tag{9}$$

$$\hat{G}_{ij} = \mathrm{i}\cdot[\hat{X},\hat{P}]_{ij} = \mathrm{i}\left(\hat{X}_{ik}\hat{P}_{kj} - \hat{X}_{kj}\hat{P}_{ik}\right) \tag{10}$$

Here, [,] is the commutator as an $N\times N$ matrix. For SU(N), the ordering ambiguity as an operator acting on Hilbert space does not exist.

1.2 Wave packets

We are interested in low-energy wave packets satisfying

$$\langle\Psi|\,\hat{X}_I\,|\Psi\rangle = Y_I\,, \qquad \langle\Psi|\,\hat{P}_I\,|\Psi\rangle = Q_I\,. \tag{11}$$

Equivalently,

$$\int \mathrm{d}\vec{x}\; x_i\psi^2(\vec{x}) = y_i \tag{12}$$

$$\int \mathrm{d}\vec{x}\psi^2 \frac{\partial\varphi}{\partial x_i} = q_i\,. \tag{13}$$

In terms of the BNAF,

$$\int \mathrm{d}\vec{z}\; p_0(\vec{z}) x_{\theta,i}(\vec{z}) = \mathbb{E}_{\vec{z}\sim p_0(\vec{z})}\left[x_{\theta,i}(\vec{z})\right] = y_i\,, \tag{14}$$

$$\int \mathrm{d}\vec{z}\; p_0(\vec{z}) J^{-1}_{\theta,ij}(\vec{z}) \frac{\partial\varphi(\vec{x}(\vec{z}))}{\partial z_i} = \mathbb{E}_{\vec{z}\sim p_0(\vec{z})}\left[\frac{\partial\varphi(\vec{x}_\theta(\vec{z}))}{\partial x_{\theta,i}}\right] = q_i\,. \tag{15}$$

A simple way to impose these constraints is to add a soft constraint term

$$c_{y,q}\left(\mathrm{Tr}\left(\langle\Psi|\,\hat{X}_I\,|\Psi\rangle - Y_I\right)^2 + \mathrm{Tr}\left(\langle\Psi|\,\hat{P}_I\,|\Psi\rangle - Q_I\right)^2\right) \tag{16}$$

to the cost function. During the training, we estimate $\langle\Psi|\,\hat{X}_I\,|\Psi\rangle$ and $\langle\Psi|\,\hat{P}_I\,|\Psi\rangle$ by the average in each batch. We need to take the batch size sufficiently large to get accurate estimate.

1.2.1 Batch and epoch

We need to approximate the integral over $\vec{z}$ by average over a finite number of samples. In terms of ML, one average over *batch size* is obtained in each *batch*. Because each sample is completely random, there is no mandatory reason to introduce the notion of *epoch*, but we still use this word to call a set of batches and record the energy etc only for the last batch of each epoch.

1.3 PyTorch implementation

The code can be found at `https://github.com/masanorihanada/VMC_Bosonic_Matrix_Model`.

1.3.1 Neural network for ψ_θ

$f_\theta : \vec{z} \to \vec{x}_\theta$ is given in `BNAF.py`.

`get_weight_mask` gives a mask to realize the BNAF architecture. Specifically, diagonal blocks and off-diagonal blocks are treated separately. This is because, in `MaskedLinear`, diagonal and off-diagonal blocks are treated differently. Specifically, the masked weights are defined by

```
masked_weight = torch.exp(self.weight) * self.mask_d + self.weight * self.mask_o
```

The ones for the diagonal block are e^w while the ones for off-diagonal blocks are w. With this choice, the diagonal part does not becomes zero. This property is important for the normalized flow.

`log_Jacobian` calculates Jacobian matrix J_{ij} and $\log J$ ($J \equiv \det J_{ij}$). Note that $\log J$ is related to ψ by (6). A tricky technical issue is that `torch.autograd.functional.jacobian`, which be used to obtain Jacobian very easily, destroys computational graph. Therefore, we used `torch.autograd.functional.jvp` and wrote a bit length routine.

1.3.2 Neural network for $\varphi_{\theta'}$

We use a single-hidden-layer dense neural network with leaky ReLU activation for the hidden layer and no activation for the output, which takes $\vec{x}$ as the input (not $\vec{z}$), i.e.,

$$\varphi_{\theta'} : \vec{x} \to \vec{a} = W\vec{x} + \vec{b} \to \vec{x}' = f(\vec{a}) \to W'\vec{x}' + b' \in \mathbb{R}\,. \tag{17}$$

The number of nodes in the hidden layer is taken to be α' times number of inputs. This is defined in `NN_phase_factor.py`. To compute $\partial_x \varphi$, we can use `torch.autograd.grad`; see `center_of_wave_packet` function in `observables.py`.

2 Introducing fermions

To introduce fermions, we write a generic quantum state in the following form:

$$\int d\vec{x}\psi(\vec{x})\,|g(\vec{x})\rangle\ . \tag{18}$$

Here, $\psi(\vec{x}) \geq 0$, and $|g(\vec{x})\rangle$ is a unit vector

$$|g(\vec{x})\rangle = \sum_{\vec{n}} g_{\vec{n}} e^{\mathrm{i}\varphi_{\vec{n}}}(\vec{x})\,|\vec{n}\rangle\ , \qquad \sum_{\vec{n}} g_{\vec{n}}(\vec{x})^2 = 1 \tag{19}$$

and

$$g_{\vec{n}} \geq 0\,, \qquad \varphi_{\vec{n}} \in \mathbb{R}\,. \tag{20}$$

where $\vec{n} = (n_1, n_2, \cdots, n_K)$, $n_i = 0$ or 1, and $\vec{n}$ is the Fock states corresponding to fermion excitation level $\vec{n}$.

For a fixed $\vec{x}$, we can regard $\rho(\vec{n}|\vec{x}) \equiv (g_{\vec{n}}(\vec{x}))^2$ as a probability distribution. We can generate $\vec{n}$ autoregressively by using the chain rule of conditional probability,

$$\begin{aligned}\rho(\vec{n}|\vec{x}) &= \rho(n_1, n_2, \cdots, n_K|\vec{x}) \\ &= \rho(n_1|\vec{x}) \cdot \rho(n_2|n_1, \vec{x}) \cdot \rho(n_3|n_1, n_2, \vec{x}) \cdots \rho(n_K|n_1, n_2, \cdots, n_{K-1}, \vec{x}) \,. \end{aligned} \tag{21}$$

ψ_θ is constructed as before, and we calculate $\langle \Psi_\theta | \hat{H} | \Psi_\theta \rangle$ by reweighting from the probability distribution $\psi_\theta^2(\vec{x})(g_{\vec{n}}(\vec{x}))^2$.

Noticing that

$$\langle \vec{n}| \hat{H} |\vec{n}'\rangle = \left(-\frac{1}{2}\partial_{\vec{x}}^2 + V(\vec{x})\right)\delta_{\vec{n}\vec{n}'} + \Phi_{\vec{n}\vec{n}'}(\vec{x}) \,, \tag{22}$$

where $\Phi_{\vec{n}\vec{n}'}(X)$ comes from the fermion part of $\hat{H}$, we can write the energy as

$$\langle \Psi| \hat{H} |\Psi\rangle = \sum_{\vec{n}} \left\langle \tilde{\Psi}_{\vec{n}} \right| \hat{H} \left| \tilde{\Psi}_{\vec{n}} \right\rangle + \sum_{\vec{n},\vec{n}'} \int d\vec{x}\ \psi^2(\vec{x}) g_{\vec{n}}(\vec{x}) g_{\vec{n}'}(\vec{x}) e^{\mathrm{i}(\varphi_{\vec{n}}(\vec{x}) - \varphi_{\vec{n}'}(\vec{x}))} \Phi_{\vec{n}\vec{n}'}(\vec{x}) \,, \tag{23}$$

$$\left|\tilde{\Psi}_{\vec{n}}\right\rangle = \int d\vec{x}\ \psi(\vec{x}) g_{\vec{n}}(\vec{x}) e^{\mathrm{i}\varphi_{\vec{n}}(\vec{x})} |\vec{x}\rangle \ . \tag{24}$$

For numerical computations via NN, we use the following relations:

$$\begin{aligned}\sum_{\vec{n}} \left\langle \tilde{\Psi}_{\vec{n}} \right| \hat{H} \left| \tilde{\Psi}_{\vec{n}} \right\rangle &= \sum_{\vec{n}} d\vec{x}\ \psi^2 g_{\vec{n}}^2 \left(\frac{1}{2}(\partial_{\vec{x}} \log(\psi g_{\vec{n}}))^2 + \frac{1}{2}(\partial_{\vec{x}} \varphi_{\vec{n}})^2 \right) \\ &= \sum_{\vec{n}} d\vec{z}\ g_{\vec{n}}^2 \left(\frac{1}{2}(\partial_{\vec{x}} \log(\psi g_{\vec{n}}))^2 + \frac{1}{2}(\partial_{\vec{x}} \varphi_{\vec{n}})^2 \right) \\ &= \mathbb{E}_{\vec{z}\sim p_0(\vec{z}),\ \vec{n}\sim\rho(\vec{n}|\vec{x}_\theta(\vec{z}))} \left[\frac{1}{2}(\partial_{\vec{x}} \log(\psi g_{\vec{n}}))^2 + \frac{1}{2}(\partial_{\vec{x}} \varphi_{\vec{n}})^2 \right] . \end{aligned} \tag{25}$$

$$\begin{aligned}&\sum_{\vec{n},\vec{n}'} \int d\vec{x}\ \psi^2(\vec{x}) g_{\vec{n}}(\vec{x}) g_{\vec{n}'}(\vec{x}) e^{\mathrm{i}(\varphi_{\vec{n}}(\vec{x}) - \varphi_{\vec{n}'}(\vec{x}))} \Phi_{\vec{n}\vec{n}'}(\vec{x}) \\ &\quad = \mathbb{E}_{\vec{z}\sim p_0(\vec{z}),\ \vec{n}\sim\rho(\vec{n}|\vec{x}_\theta(\vec{z}))} \left[\sum_{\vec{n}'} \frac{g_{\vec{n}'}(\vec{x})}{g_{\vec{n}}(\vec{x})} e^{\mathrm{i}(\varphi_{\vec{n}}(\vec{x}) - \varphi_{\vec{n}'}(\vec{x}))} \Phi_{\vec{n}\vec{n}'}(\vec{x}) \right] . \end{aligned} \tag{26}$$

References

[1] Norbert Bodendorfer, Onur Oktay, Vaibhav Gautam, Masanori Hanada, and Enrico Rinaldi. Variational Monte Carlo with Neural Network Quantum States for Yang-Mills Matrix Model. 8 2024.

[2] Nicola De Cao, Ivan Titov, and Wilker Aziz. Block neural autoregressive flow, 2019.