Accelerating Green's Functions Calculations in Hubbard DQMC

James Neuhaus

May 5, 2021

Hubbard Model DQMC - Introduction

Determinate Quantum Monte Carlo (DQMC) is a commonly used stochastic method to probe the parameter space of a finite temperature quantum system. As the Monte Carlo walks through phase space, you can periodically make measurements in order to find expectation values of the system. While DQMC is a great method, this is generally computationally expensive. The Monte Carlo updates as it walks through a Markov chain with an accept/reject step whereby a weight w_{new} is compared against the previous step w_{old} and a randomized value to determine if the update happens. In order for the Monte Carlo to operate appropriately, the randomized value is scaled to ensure that roughly half of all updates are accepted. If an update is rejected, the Monte Carlo remains in its previous state.

In a DQMC where the terms of the Hamiltonian do not commute, a Matsubara imaginary time τ path integral formalism is used. This method breaks down a matrix element for the Boltzman operator $e^{-\beta H}$ into smaller components.

$$\begin{aligned}\left\langle \Psi_L \,\middle|\, e^{-\beta H} \,\middle|\, \Psi_R \right\rangle = \sum_{\Psi_n} \left\langle \Psi_L \,\middle|\, e^{-\Delta\tau H} \,\middle|\, \Psi_{N-1} \right\rangle \cdots \\ \left\langle \Psi_2 \,\middle|\, e^{-\Delta\tau H} \,\middle|\, \Psi_1 \right\rangle \left\langle \Psi_1 \,\middle|\, e^{-\Delta\tau H} \,\middle|\, \Psi_R \right\rangle\end{aligned} \tag{1}$$

The partition function for the system is defined

$$\mathcal{Z} = Tr[e^{-\beta H}] = \sum_{\Psi} \left\langle \Psi | e^{-\beta H} | \Psi \middle| \Psi | e^{-\beta H} | \Psi \right\rangle \tag{2}$$

Thankfully, as we shall show later, this incalculable quantity is not necessary for DQMC, as it appears in both the numerator and denominator in our update step.

It is important keep the parameter $\Delta\tau$ small, as we rely on the Trotter approximation in our calculations. For n terms in the Hamiltonian the Trotter approximation becomes

$$e^{-\Delta\tau H} = e^{-\Delta\tau H_1} e^{-\Delta\tau H_2} \cdots e^{-\Delta\tau H_n} + \mathcal{O}((\Delta\tau)^2) \tag{3}$$

It is important to note that the error scales $\mathcal{O}((\Delta\tau)^2)$. Using other tricks it is possible to reduce this error to $\mathcal{O}((\Delta\tau)^3)$, but this can still become significant at insufficiently small $\Delta\tau$.

The Hubbard model is a tight-binding model defined by its Hamiltonian,

$$H^{Hub} = -t \sum_{\langle ij \rangle,\sigma} (a^{\dagger}_{i\sigma} a_{j\sigma} + h.c.) + U \sum_{i} \left(n_{i\uparrow} - \frac{1}{2}\right)\left(n_{i\downarrow} - \frac{1}{2}\right) - \mu \sum_{i\sigma} n_{i\sigma} \quad (4)$$

The first term represents the kinetic energy of the electrons, with t as a relative energy magnitude. By convention in DQMC $t = 1$. The second term contains the Coulomb repulsion between electrons on the same site, with U representing the magnitude of this interaction. Electrons at different sites are treated as fully screened, and thus do not contribute to the total energy. Finally, μ is a chemical potential. At half-filling $\mu = 0$. By scaling μ you can simulate electron or hole doping without modifying your lattice geometry and breaking symmetries.

Using the Trotter approximation we can take the exponentiated Hamiltonian and represent it as a non-interacting piece and the interacting piece

$$e^{-\Delta\tau H} \approx e^{-\Delta\tau H_1} e^{-\Delta\tau U\left(n_\uparrow - \frac{1}{2}\right)\left(n_\downarrow - \frac{1}{2}\right)} \quad (5)$$

Rather than deal with the term which is quadratic in number operators, we use a Hubbard-Stratonovich (HS) transformation, which uses the properties of Gaussian integrals to reduce the problem to linear combinations of the number operators under an integral or sum. For $U > 0$

$$e^{-\Delta\tau U\left(n_\uparrow - \frac{1}{2}\right)\left(n_\downarrow - \frac{1}{2}\right)} = \frac{1}{2} e^{-\frac{1}{4}\Delta\tau U} \sum_{h=\pm} e^{\alpha h (n_\uparrow - n_\downarrow)} \quad (6)$$

The h values are stored in an auxiliary HS field. α is a constant defined such that $\cosh\alpha = \exp(\Delta\tau |U|/2)$.

Let us discuss how determinantal weights are calculated. Intermediate matrix calculations are made on a single particle propagator B matrix with components calculated by

$$B_{ij}(\tau_2, \tau_1) = \left\langle 0 \middle| c_i \left[\mathcal{T} \exp\left(- \int_{\tau_1}^{\tau_2} H(\tau) d\tau \right) \right] c^{\dagger}_j \middle| 0 \right\rangle \quad (7)$$

Using a delta matrix $\Delta^{\sigma}(i, \tau)$ for the proposed site i to update, we get a matrix which is all zeros except for

$$\Delta^{\sigma}_{ii}(i, \tau) = e^{\sigma\alpha(x'_i(\tau) - x_i(\tau))} - 1 \quad (8)$$

We define our equal-time Green's function such that

$$G^{\sigma}(\tau, \tau) = [I + B^{\sigma}(\tau, 0) B^{\sigma}(\beta, \tau)]^{-1} \quad (9)$$

But since the equal-time Green's function is updated through this formula

$$G^{\sigma}(\tau', \tau') = B^{\sigma}(\tau', \tau) G^{\sigma}(\tau, \tau) B^{\sigma}(\tau', \tau)^{-1} \quad (10)$$

and the ratio of weights is

$$\mathcal{R}^{\sigma} \equiv \frac{w^{\sigma}_{new}}{w^{\sigma}_{old}} = \det[I + \Delta^{\sigma}(i,\tau)(I - G^{\sigma}(\tau,\tau))] \tag{11}$$

and the final ratio for updating is $\mathcal{R} = \mathcal{R}^{\uparrow}\mathcal{R}^{\downarrow}$.

For much greater depth and the full derivation, see Quantum Monte Carlo Methods [1].

Motivation and Data Set

The amount of wall-time that goes into calculating Green's functions is significant. Depending on the Monte Carlo parameters, Hubbard model parameters, and available hardware these calculations can utilize 30-60% of wall-time. On major problem is the calculations multiply and exponentiate fairly stiff matrices. Even with double precision floating point, numerical error can accumulate very quickly. Thus, much of the calculation time comes from utilizing numerical methods to properly condition matrices. Typically, matrices must have calculations performed on columns of decreasing eigenvalue. Additionally, most DQMC algorithms utilize costly QR factorization when calculating the equal-time Green's functions.

In this project I have attempted to determine if it is possible to estimate the equal-time Green's functions using machine learning techniques. The "old" Green's functions for the previous L imaginary time-steps are fed into a neural net and the output were the new diagonal elements of the Green's function for τ'. L is defined as by $L = \beta/\Delta\tau$. Being able to estimate the Green's functions could accelerate the reject step by throwing out values which would be obvious rejects, and limit calculation of the true Green's functions on updates likely to be accepted.

After modifying the code to output a CSV of the needed Green's function values, I ran Dr. Steven Johnston's CPU-based Hubbard-Holstein DQMC code to generate data [2]. All Holstein model parameters were set to zero, which for this code made it a pure Hubbard model DQMC. I utilized a 2x2 2D lattice, with $t = 1$, $U = 6$, $\beta = 1$, and $\Delta\tau = 0.1$. Old and new Green's functions were output to a CSV file for 2,000,000 calculations. Since the DQMC code is Fortran based, I deemed it prudent to dump that data to a file rather than train on the fly by integrating C-based TensorFlow methods.

Model, Implentation, Testing

My initial intent was to use a convolutional neural network (CNN). However, early testing showed that a 3D CNN using Keras was not feasible and a deep neural net (DNN) would give better results. The periodic boundary conditions could not be implemented using Keras' built in padding capability. While I could eventually build a CNN from scratch using TensorFlow, this is beyond

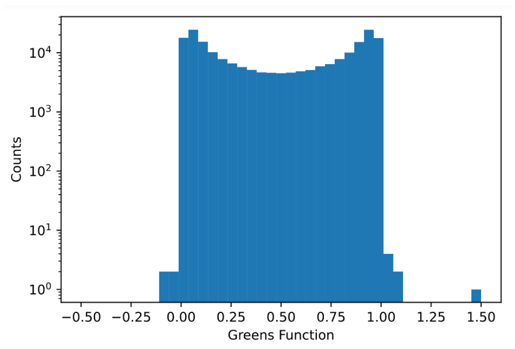

Figure 1: Greens functions spectrum from DQMC code (log y)

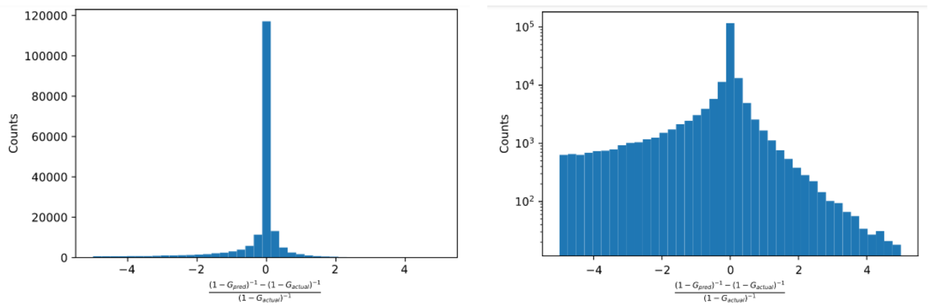

Figure 2: Difference between calculated and estimated (DNN) Green's functions (Left: linear, Right: log y)

the scope of this project. Because of the small lattice size, the shape of the input tensor was $N_x \times N_y \times L$ which for our lattice size and number of time steps made an input tensor shape of $2 \times 2 \times 10$. Our convolutional 3D layer was limited to a kernel size of 2, with boundary conditions greatly affecting the outcome. This network architecture may be worth revisiting for larger lattices, or with a recurring boundary condition implementation.

After switching to a fully connected DNN, I initially made a common, but major mistake. My output layer initially used a ReLu activation. Considering Green's functions can be negative, I soon swapped to a linear activation. I tested sigmoid and ReLu activations for my dense layers, as well as a variation of neurons. Ultimately I settled on a DNN of $40 \times 160 \times 80 \times 40 \times 4$ with a 10% dropout between layers. Attempts to tweak beyond this led to similar results.

Results

Figure 1 shows the range of values of Green's functions calculated by the DQMC code. Generally, these fell in the range of $(0, 1)$. However, this was not exact, and there was no explicit upper or lower bound. Thus, I could not pre-process my data in a way to guarantee it would fall in this range. Thus, while I could have pre-processed this data set to fall in the range of a sigmoid, there would be no guarantee that this was generally applicable for all future DQMC configurations. Further investigation is needed to test full feasibility of discarding values outside this range or scaling to a fixed range.

The difference in between the DQMC calculated values and the DNN estimated Green's functions was a Gaussian distribution centered on the origin. This distribution had a standard deviation was calculated to be 0.024.

In Figure 2 we view the Green's functions from DQMC and DNN as transformed to their relevant values for the accept/reject step, $(1 - G_{ii})^{-1}$. Here we see that in the left linear plot there is a very good agreement between the values. In the right log plot there appears to be some disagreement where the DNN predicted value tends to be lower than the actual value.

Discussion

Ultimately, my attempts to calculate Green's functions in this manner were somewhat successful. The Green's functions themselves are very unstable, as their value tends to run in the range of $(0, 1)$. Considering the update step in our Monte Carlo code relies on a factor of $(1 - G_{ii}(\tau, \tau))^{-1}$, pre processing the data may be an advisable next step.

While using a smaller lattice size of 2×2 seemed like a good idea for the sake of an initial test, doing so resulted in preventing the use of a 3D CNN, since the number of imaginary time slices was much larger than either lattice edge.

Using a pre-existing Monte Carlo [2] made code development much simpler, but cost me the ability to translate from the Green's function calculation to the accept/reject step directly without significantly modifying Dr. Johnston's code.

The result of this work is inconclusive, but shows significant promise in using machine learning techniques to accelerate Hubbard Model Monte Carlos. This author was too engrossed in dealing with the technical aspect, and in the future would like to develop better metrics for defining success while retrying these techniques.

References

[1] J. Gubernatis, N. Kawashima, and P. Werner, *Quantum monte carlo methods: algorithms for lattice models* (Cambridge University Press, 2016) (cited on page 3).

[2] S. Johnston, E. A. Nowadnick, Y. F. Kung, B. Moritz, R. T. Scalettar, and T. P. Devereaux, “Determinant quantum monte carlo study of the two-dimensional single-band hubbard-holstein model”, Phys. Rev. B **87**, 235133 (2013) (cited on pages 3, 5).