StableDQMC.jl: Fast and stable determinant quantum Monte Carlo

Carsten Bauer[1]

[1]*Institute for Theoretical Physics, University of Cologne, 50937 Cologne, Germany*

(Dated: February 25, 2020)

In these notes we assess numerical stabilization methods employed in fermion many-body quantum Monte Carlo simulations. In particular, we review the origin of numerical instabilities in the determinant quantum Monte Carlo (DQMC) framework, arising in the calculation of equal-time and time-displaced Green's functions, and empirically compare various matrix decomposition and inversion schemes to gain control over computations. Besides numerical accuracy we also benchmark the computational efficiency of the different stabilization methods. We identify a scheme based on QR decompositions which is both stable and fast and therefore well suited for DQMC applications. Compared to approaches relying on singular value decompositions we find that our QR is superior. We use the Julia programming language for our assessment and provide implementations of all discussed techniques in the open-source software library `StableDQMC.jl`.

I. INTRODUCTION

Many-fermion systems play an important role in condensed matter physics. Due to their intrinsic correlations they feature rich phase diagrams which can not be captured by purely classical nor non-interacting theories. Especially at the lowest temperatures, quantum mechanical fluctuations driven by Heisenberg's uncertainty principle become relevant and lead to novel states of matter like superconductivity or ones beyond the Fermi liquid paradigm. Because of the presence of interactions, predicting microscopic and thermodynamic properties of fermion many-body systems is inherently difficult. Analytical approaches are typically doomed to fail in cases where one can not rely on the smallness of an expansion parameter.

Fortunately, the determinant quantum Monte Carlo (DQMC) method overcomes this limitation. The key feature of DQMC is that it is numerically exact - given sufficient computation time the systematical error is arbitrarily small. Provided the absence of the famous sign-problem, it allows us to efficiently explore the relevant region of the exponentially large configuration space in polynomial time. It is an important unbiased technique for obtaining reliable insights into the physics of many-fermion systems.

Although conceptually straightforward, care has to be taken in implementing DQMC due to inherent numerical instabilities. It is the purpose of this work to review stabilization schemes to heal those algebraic issues and to compare them with respect to accuracy and speed. Specifically, the structure of the paper is as follows. We start by providing a brief introduction into the DQMC method in Sec. II. In Sec. III we illustrate numerical instabilities arising in the DQMC formalism and recall their origin. Following this, we present (Sec. IV) and benchmark (Sec. V) different numerical stabilization schemes in the context of the computation of the equal-times Green's function and its determinant. Lastly, we turn to the calculation of the time-displaced Green's function in Sec. VI before concluding and summarizing in Sec. VII.

We provide implementations of all discussed algorithms, as well as the code to recreate all the plots of these notes, in form of the Julia package `StableDQMC.jl`.

II. QUANTUM MONTE CARLO

We begin by recalling the determinant - or auxiliary field - quantum Monte Carlo (DQMC) algorithm [1] for a generic quantum field theory that can be split into a purely bosonic part S_B and a part S_F. The latter comprises fermion kinetics T and boson-fermion interactions V. An example is the famous Hubbard model after decoupling the on-site interaction $Un_{i,\uparrow}n_{i,\downarrow}$ by means of a Hubbard-Stratonovich or Hirsch transformation in either the spin or charge channel [2]. As per usual, the central quantity of interest is the partition function

$$\mathcal{Z} = \int D\left(\psi, \psi^\dagger, \phi\right) e^{-S_B - S_F} . \tag{1}$$

The basic idea of DQMC is to switch from the d dimensional quantum theory to a $D = d + 1$ dimensional classical theory. The extra finite dimension of the classical theory is imaginary time τ. It has a length proportional to the inverse temperature $\beta = 1/T$ and is discretized into M time slices, $\beta = M\Delta\tau$. Applying a Trotter-Suzuki decomposition [3, 4] one obtains

$$\mathcal{Z} = \int D\phi\, e^{-S_B} \mathrm{Tr}\left[\exp\left(-\Delta\tau \sum_{l=1}^{M} \psi^\dagger \left[T + V_\phi\right] \psi\right)\right] . \tag{2}$$

Next, the exponential is separated which leads to a systematic error of the order $\mathcal{O}\left(\Delta\tau^2\right)$,

$$\begin{aligned} e^{A+B} &\approx e^A e^B \\ e^{-\Delta\tau(T+V)} &\approx e^{-\frac{\Delta\tau}{2}T} e^{-\Delta\tau V} e^{-\frac{\Delta\tau}{2}T} + \mathcal{O}\left(\Delta\tau^3\right), \\ \mathcal{Z} &= \int D\phi\, e^{-S_B} \mathrm{Tr}\left[\prod_{l=1}^{m} B_l\right] + \mathcal{O}\left(\Delta\tau^2\right) . \end{aligned} \tag{3}$$

Here, $B_l = e^{-\frac{\Delta\tau}{2}\psi^\dagger T \psi} e^{-\Delta\tau \psi^\dagger V_\phi \psi} e^{-\frac{\Delta\tau}{2}\psi^\dagger T \psi}$ are imaginary time slice propagators. Note that their potential contribution $e^{-\Delta\tau\psi^\dagger V_\phi \psi}$ depends on the boson ϕ due to fermion-boson coupling. Rewriting the trace in (3) as a determinant, an identity which can be proven CITE, yields the fundamental form

$$\mathcal{Z} = \int D\phi\, e^{-S_B} \det G_\phi^{-1} + \mathcal{O}\left(\Delta\tau^2\right), \tag{4}$$

where

$$G = (\mathbb{1} + B_M B_{M-1} \cdots B_1)^{-1} \tag{5}$$

is the equal-time Green's function of the system.

As per Eq. (4), the probability weight appearing in a Metropolis Monte Carlo scheme reads

$$p = \min\left\{1, e^{-\Delta S_\phi} \frac{\det G}{\det G'}\right\}, , \tag{6}$$

which tells us that, considering a generic update, we need to compute the Green's function G and it's determinant for both the current and the proposed state (G') of the Markov walker. For local updates, however, one can typically avoid those costly calculations and rather compute the ratio of determinants in Eq. (6) directly.

Importantly, it is only under specific circumstances, such as the presence of a symmetry, that the integral kernel can be safely interpreted as a probability weight as G_ϕ and its determinant are generally complex valued. This is the famous sign problemCITE.

III. NUMERICAL INSTABILITIES

To showcase the typical numerical instabilities arising in the DQMC framework we consider the Hubbard model in one dimension at half filling,

$$H = -t \sum_{\langle i,j \rangle} c_i^\dagger c_j + U \sum_i \left(n_{i\uparrow} - \frac{1}{2}\right)\left(n_{i\downarrow} - \frac{1}{2}\right), \tag{7}$$

and set the hopping amplitude to unity, $t = 1$.

As seen from Eq. (5), the building block of the equal-time Green's function is the slice matrix product chain

$$B(\beta, 0) \equiv B_M B_{M-1} \cdots B_1 = \underbrace{B B \cdots B}_{M \text{ factors}}. \tag{8}$$

where in the second equality we have assumed that the slice matrices are independent of imaginary time to simplify our numerical analysis.

First, we consider the non-interacting system, $U = 0$. In Fig. 1, we show that a naive computation of Eq. 8 is doomed to fail for $\beta \geq \beta_c \approx 10$. Leaving a discussion of the stabilization of the computation for the next section, let us highlight the origin of this instability. The eigenvalues of the system are given by

$$\epsilon_k = -2t \cos(k). \tag{9}$$

The energy values are bounded by $-2t \leq \epsilon_k \leq 2t$. Hence, a single positive definite slice matrix $B = e^{-\Delta\tau T}$ has a condition number of about $\kappa \approx e^{4|t|\Delta\tau}$, which gives $\kappa \approx e^{4|t|M\Delta\tau} = e^{4|t|\beta}$ for the product chain $B(\tau, 0)$. This implies that the scales present in $B(\tau, 0)$ broaden exponentially at low temperatures $T = 1/\beta$ and roundoff errors due to finite machine precision will spoil a naive computation. We can

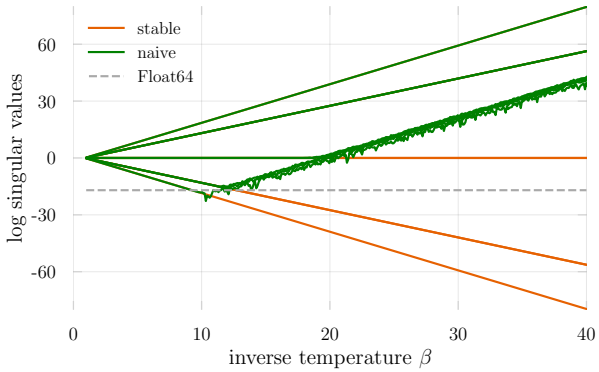

FIG. 1. **Numerical instabilities** (green) due to finite machine precision (`Float64`) in the calculation of the slice matrix product chain $B_M B_{M-1} \cdots B_1$ for model (7).

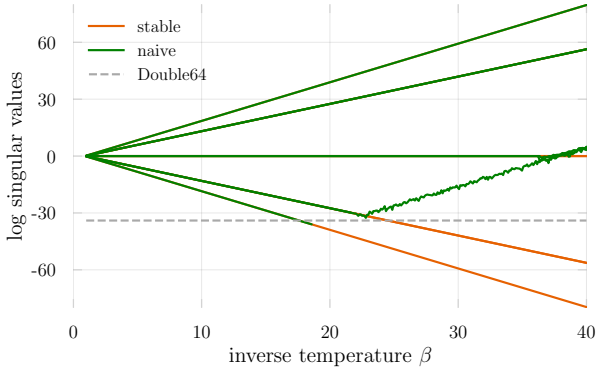

FIG. 2. **Numerical instabilities** due to finite machine precision (`Double64`) in the calculation of the slice matrix product chain $B_M B_{M-1} \cdots B_1$ for model (7).

estimate the inverse temperature of this breakdown of the calculation for the data type `Float64`, that is double floating-point precision [5], by solving $\kappa(\beta) \sim 10^{-17}$ for β_c. This gives $\beta_c \approx 10$ in good agreement with what we observe in Fig. 1. Switching to the non-IEEE data type `Double64`, we see in Fig. 2 that the onset of roundoff errors is shifted to lower temperatures, in accordance with expectations.

Another consequence of these numerical imprecisions is that the $B(\tau, 0)$ obtained from a naive computation are generally not invertible and the inversion in Eq. 5 is ill defined. This clearly prohibits a safe calculation of the equal-time Green's function and asks for more sophisticated techniques.

IV. STABILIZATION

A trivial solution to the issue outlined above is to perform all numerical operations with arbitrary precision. In Julia this can be done by using the `BigFloat` data type. However, this comes at the expense of (unacceptable) slow performance due to algorithmic overhead and lack of hardware support. Arbitrary precision numerics is nevertheless a valuable tool and we

will use it to benchmark the accuracy of stabilization methods below[6].

How can we get a handle on the numerical instabilities in a floating point precision computation? The idea is to keep the broadly different scales separated throughout the computation (as much as possible) and only mix them in the final step, if necessary. A useful tool along these lines are matrix decompositions,

$$B = UDX. \tag{10}$$

Here, U and X are matrices containing scales of the order of unity and D is a real diagonal matrix with the broad range of scales of B separated on the diagonal. We will refer to the values in D as singular values independent of the particular decomposition.

Instead of calculating products B_2B_1 appearing in $B(\tau, 0)$, Eq. 8, directly, we utilize Eq. 10 to define a stable matrix multiplication (`fact_mult` in `StableDQMC.jl`)

$$\begin{aligned} B_2B_1 &= \underbrace{U_2D_2X_2}_{B_2}\underbrace{U_1D_1X_1}_{B_1} \\ &= U_2\,\underbrace{(D_2((X_2U_1)D_1))}_{U'D'X'}\,X_1) \\ &= U_rD_rX_r. \end{aligned} \tag{11}$$

Here, $U_r = U_2U'$, $D_r = D'$, $X_r = X'X_1$, and $U'D'X'$ indicates an intermediate matrix decomposition. If we follow this scheme, in which parentheses indicate the order of operations, largely different scales present in the diagonal matrices won't be mixed throughout the computation. Repeating this procedure, we obtain a numerically accurate UDX decomposition of the full slice matrix product chain $B(\tau, 0)$.[7] We note in passing that in a practical DQMC implementation it is often unnecessary to stabilize every single matrix product but. Instead one typically performs a mixture of naive and stabilized products for the sake of speed while still retaining numerical accuracy [8].

A. Equal-time Green's function

Looking at the equal-time Green's function in Eq. 5, we have to be careful to keep scales separated in the inversion of $1 + B(\beta, 0)$ as well. In fact, small singular values of the order of unity in $B(\beta, 0)$ would even be washed out just by naively adding the identity matrix alone. Fortunately, these issues can be circumvented as well.

A straightforward procedure (`inv_one_plus`) to add the unit matrix and perform the inversion in a stabilized manner is given by [8, 9]

$$\begin{aligned} G &= [\mathbb{1} + UDX]^{-1} \\ &= [U\,\underbrace{(U^\dagger X^{-1} + D)}_{udx}\,X]^{-1} \\ &= [(Uu)d(xX)]^{-1} \\ &= U_rD_rX_r, \end{aligned} \tag{12}$$

with $U_r = (xX)^{-1}$, $D_r = d^{-1}$, $X_r = (Uu)^{-1}$.

Another prescription for a stabilized inversion (`inv_one_plus_loh`), where we initially separate the scales in as $D_p = \max(D, 1)$ and $D_m = \min(D, 1)$ and perform two intermediate decompositions, is given by [10, 11]

$$\begin{aligned} G &= [\mathbb{1} + UDX]^{-1} \\ &= [\mathbb{1} + UD_mD_pX]^{-1} \\ &= [(X^{-1}D_p^{-1} + UD_m)D_pX]^{-1} \\ &= X^{-1}\,\underbrace{[D_p^{-1}\underbrace{(X^{-1}D_p^{-1} + UD_m)}_{udx})^{-1}]}_{udx} \\ &= U_rD_rX_r, \end{aligned} \tag{13}$$

with $U_r = X^{-1}u$, $D_r = d$, and $X_r = x$. We will demonstrate below that it is sometimes necessary to employ this second procedure to obtain accurate results for G.

So far we haven't specified a concrete decomposition $B = UDX$. In fact, there are a couple of choices, two of which we will focus on in what follows.

1. *SVD ($UDV^\dagger$)*

A SVD is given by

$$B = USV^\dagger, \tag{14}$$

where U is unitary, S is a real diagonal matrix, and $V^\dagger$ is unitary. In this case we can use the unitarity of U and $V^\dagger$ to calculate inverse terms like, for example, $(Uu)^{-1}$ in the last line of 12 as $(Uu)^{-1} = u^\dagger U^\dagger$, which is generally cheaper.

Julia offers a couple of purely-Julia SVD implementations, like GenericSVD.jl, which we will use for `BigFloat` computations. However, some of the most optimized algorithms are part of LAPACK [12] and Julia defaults to those algorithms for regular floating point types. Concretely, there are three SVD functions[13] implementing different algorithms for calculating the SVD:

- `gesdd` (default): Divide-and-conquer (D&C)
- `gesvd`: Conventional
- `gesvj`: Jacobi algorithm (through JacobiSVD.jl)

which can be readily accessed via convenience wrappers of the same name exported by `StableDQMC.jl`. We will compare all of them below.

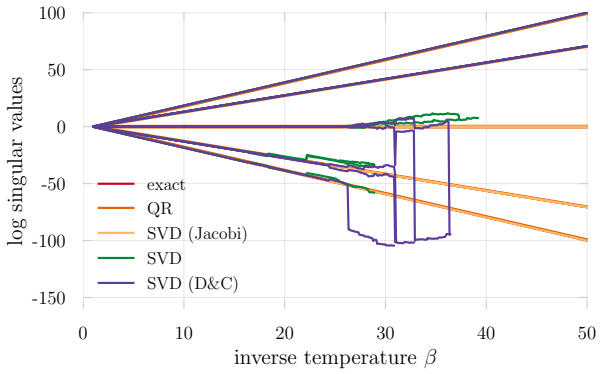

FIG. 3. **Comparison of matrix decompositions** to heal the numerical instabilities in the calculation of the slice matrix product chain $B_M B_{M-1} \cdots B_1$ for model (7). The QR and Jacobi SVD singular values seem to lie on top of the exact ones whereas regular SVD and divide-and-conquer SVD show large deviations at low temperatures $\beta \gtrsim 25$ ($\Delta\tau = 0.1$).

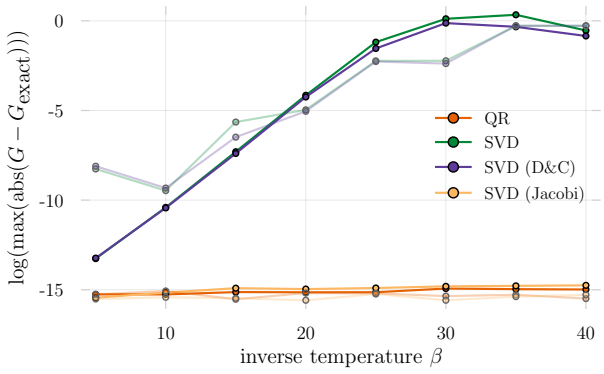

FIG. 4. **Accuracy of the Green's function** obtained from stabilized computations using the listed matrix decompositions and the inversion scheme `inv_one_plus`, Eq. (12). Shown are results for $U = 0$ (solid) and $U = 1$ (alpha transparent).

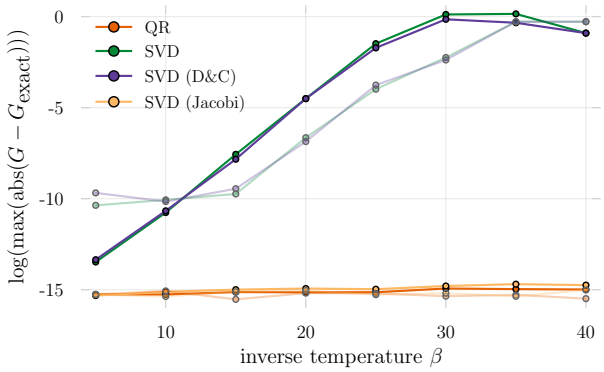

FIG. 5. **Accuracy of the Green's function** obtained from stabilized computations using the listed matrix decompositions and the careful inversion scheme `inv_one_plus_loh`, Eq. 13. Shown are results for $U = 0$ (solid) and $U = 1$ (alpha transparent).

2. *QR (UDT)*

A QR decomposition reads

$$B = QR = UDT, \tag{15}$$

where we have split R into a diagonal, D, and upper triangular piece T. Hence, $U = Q$ is unitary, $D = \mathrm{diag}(R)$ is a real diagonal matrix, and T is upper triangular. In Julia, one can obtain the QR factored form of a matrix by calling the function `qr` from the standard library `LinearAlgebra`. Analogously, a decomposition into UDT form is provided by `udt` and `udt!` in `StableDQMC.jl`.

V. BENCHMARKS

In the following we want to assess how the mentioned matrix decompositions perform in stabilized computations of $B(\beta, 0)$, the Green's function G, and its determinant $\det G$, both with respect to accuracy and speed. All results are for the Hubbard model, Eq. 7, with $U = 0$ and $U = 1$ (alpha transparent in all plots)

A. Accuracy

Before benchmarking the efficiency of an algorithm, it is crucial to check it's correctness first. Fig. 3 shows the log singular values of the slice matrix product chain $B(\beta, 0)$ stabilized with different matrix decompositions as a function of inverse temperature β. While QR and Jacobi SVD seem to lie on top of the numerically exact result, we observe large deviations for the simple and D&C SVD algorithms at low temperatues ($\beta \gtrsim 25$).[14]

Turning to the equal-time Green's function, Eq. 5, we take the results for the slice matrix chains and perform the inversions according to the schemes presented above. We take the maximum absolute difference between the obtained Green's functions and the exact G as an accuracy measure. The findings for the simple inversion scheme `inv_one_plus`, Eq. 12, are shown in Fig. 4. At high temperatures and for $U = 0$, all decompositions give the correct Green's function up to some limit close to floating point precision. However, at low temperatures only the QR decomposition and the Jacobi SVD reproduce G_{exact} reliably. They have the highest accuracy by a large margin, followed by the other SVD variants, which fail to reproduce the exact result accurately. As displayed in Fig. 5, switching to the more careful procedure `inv_one_plus_loh`, Eq. 13, does generally improve the accuracy but the deviations seen for the regular and D&C SVD schemes are still of order unity at low temperatures.

In Figs. 6, 7 we show the logarithm of the relative error of the Green's function determinant, relevant in the Metropolis acceptance[15], obtained for all combinations of matrix decompositions and inversion schemes. Both the QR decom-

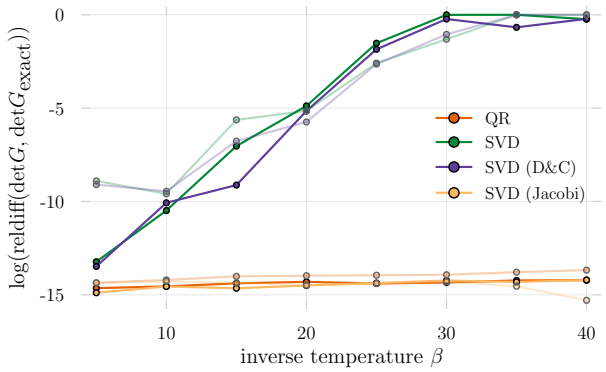

FIG. 6. **Accuracy of the determinant** of the equal-time Green's function obtained from stabilized computations using the listed matrix decompositions and the inversion scheme `inv_one_plus`, Eq. 12. Shown are results for $U = 0$ (solid) and $U = 1$ (alpha transparent).

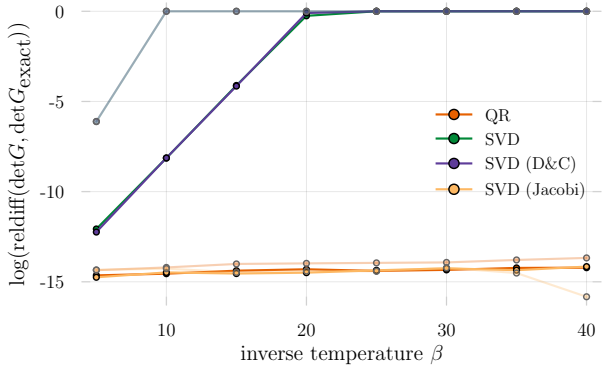

FIG. 7. **Accuracy of the determinant** of the equal-time Green's function obtained from stabilized computations using the listed matrix decompositions and `inv_one_plus_loh`, Eq. 13. Shown are results for $U = 0$ (solid) and $U = 1$ (alpha transparent).

position and the Jacobi SVD lead to accurate results for all accessed temperatures, irrespective of the employed inversion scheme. The other two SVD based methods on the other hand show large relative deviations for both `inv_one_plus` and `inv_one_plus_loh`.

These findings suggest that only the QR decomposition and the Jacobi SVD are suited for computing both the equal time Green's function and it's determinant reliably, irrespective of the inversion procedure.

B. Efficiency

Independent of the deployed inversion scheme, matrix decompositions account for most of the time cost of the Green's function calculation. Fig. 8 illustrates the raw efficiency of all SVDs relative to the QR decomposition. While the conventional SVD and the Jacobi SVD are about an order of magnitude slower, the divide-and-conquer based SVD is in the same ballpark as the QR decomposition. The Jacobi SVD variant

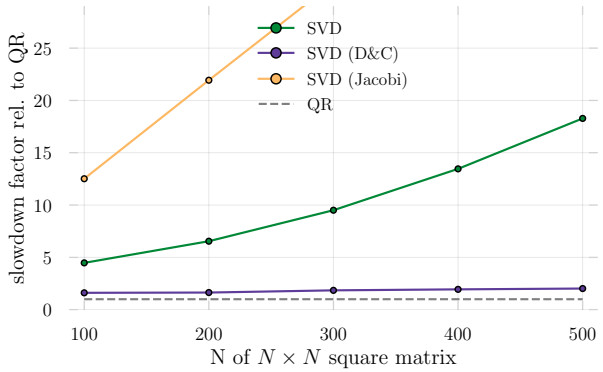

FIG. 8. **Efficiency of different matrix decompositions**. Shown are the slowdown factors of single SVDs relative to a QR decomposition of a complex matrix of size $N \times N$.

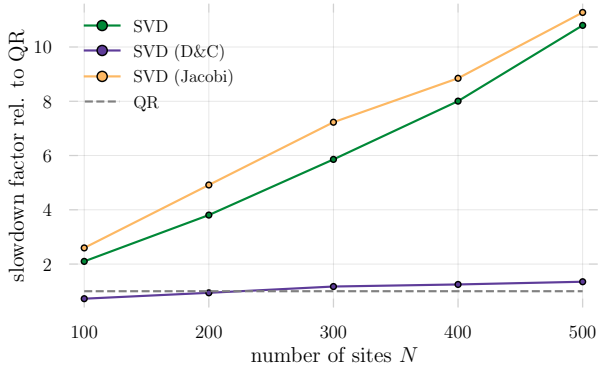

FIG. 9. **Efficiency of the stabilized Green's function calculation** using the listed matrix decompositions and the inversion scheme `inv_one_plus`, Eq. (12).

is, by far, the most costly of all considered matrix decompositions, being 10 times more time consuming than the QR decomposition, even for small system sizes.

Since the decompositions represent the performance bottleneck, we expect that these speed differences propagate and dominate benchmarks of the full Green's function computations. As visible in Figs. 4 and 5, this anticipation is qualitatively confirmed up to numerical deviations. Independent of the deployed inversion scheme, the divide-and-conquer SVD can compete with the QR decomposition in terms of speed whereas the other SVD algorithms unambiguously fall behind. We note that the relative slowdown factor is larger for the scheme by Loh *et al.*, which is understood from the fact that it requires two intermediate matrix decompositions rather than one.

VI. TIME-DISPLACED GREEN'S FUNCTION

We generalize our definition of the equal times Green's function, Eq. 5, to include the imaginary time $\tau = l\Delta\tau$ de-

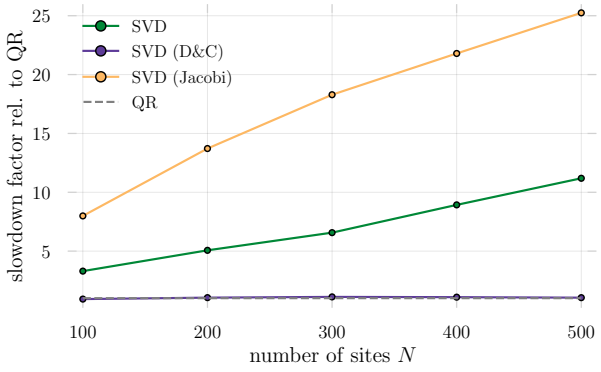

FIG. 10. **Efficiency of the stabilized Green's function calculation** using the listed matrix decompositions and the inversion scheme `inv_one_plus_loh`, Eq. (13). Shown are results for $U = 0$.

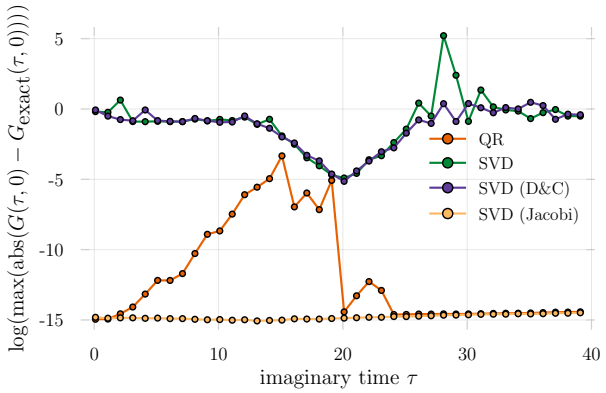

FIG. 11. **Accuracy of the time-displaced Green's function** obtained from stabilized computations using the listed matrix decompositions and the inversion scheme `inv_sum`, Eq. (19), for $\beta = 40$.

pendence,

$$G(\tau) = \langle c_i c_j^\dagger \rangle_{\phi_l} = (1 + B_{l-1} \dots B_1 B_M \dots B_l)^{-1}. \quad (16)$$

Note that $G \equiv G_1 = G_{M+1} = (1 + B_M \dots B_l)^{-1}$. The time displaced Green's function can now be defined as [8, 9]

$$G_{l_1,l_2} \equiv G(\tau_1, \tau_2) \equiv \langle T c_i(\tau_1) c_j^\dagger(\tau_2) \rangle_\varphi,$$

where T represents time ordering.

More explicitly this reads

$$G(\tau_1,\tau_2) = \begin{cases} B_{l_1} \cdots B_{l_2+1} G_{l_2+1}, & \tau_1 > \tau_2, \\ -(1 - G_{l_1+1}) (B_{l_2} \cdots B_{l_1+1})^{-1}, & \tau_2 > \tau_1. \end{cases} \quad (17)$$

In principle, this gives us a prescription for how to calculate $G(\tau_1, \tau_2)$ from the equal time Green's function $G(\tau)$ (which we know how to stabilize). However, when $|\tau_1 - \tau_2|$ is large a naive calculation of slice matrix product chains in Eq. 17 would be numerically unstable, as seen above. Also, by first calculating G we already mix important scale information in the last recombination step, in which we multiply $G = UDX$. We therefore rather compute the time-displaced Green's function directly as

$$G(\tau_1, \tau_2) = (U_L D_L X_L + U_R D_R X_R)^{-1}. \quad (18)$$

Similar to Sec. IV, we must be very careful to keep the involved scales separated as much as possible when performing the summation and the inversion. As a first explicit procedure, we consider a simple generalization of Eq. 12 (`inv_sum`),

$$\begin{aligned} G(\tau_1,\tau_2) &= [U_L D_L X_L + U_R D_R X_R]^{-1} \\ &= [U_L \underbrace{(D_L X_L X_R^{-1} + U_L^\dagger U_R D_R)}_{udx} X_R]^{-1} \\ &= [(U_L u) d^{-1} (x X_R)]^{-1} \\ &= U_r D_r X_r, \end{aligned} \quad (19)$$

where $U_r = (x X_R)^{-1}$, $D_r = d^{-1}$, and $X_r = (U_L u)^{-1}$.

Another scheme, analogous to Eq. 13, where we split the scales in D, is as follows (`inv_sum_loh`), [10]

$$\begin{aligned} G(\tau_1,\tau_2) &= [U_L D_L X_L + U_R D_R X_R]^{-1} \\ &= [U_L D_{Lm} D_{Lp} X_L + U_R D_{Rm} D_{Rp} X_R]^{-1} \\ &= \left[U_L D_{Lp} \underbrace{\left(\frac{D_{Lm}}{D_{Rp}} X_L X_R^{-1} + U_L^\dagger U_R \frac{D_{Rm}}{D_{Lp}} \right)}_{udx} X_R D_{Rp} \right]^{-1} \\ &= X_R^{-1} \underbrace{\frac{1}{D_{Rp}} [udx]^{-1} \frac{1}{D_{Lp}}}_{udx} U_L^\dagger \\ &= U_r D_r X_r, \end{aligned} \quad (20)$$

with $U_r = X_R^{-1} u$, $D_r = d$, and $X_r = x U_L^\dagger$.

We note in passing that Hirsch [16] has proposed an alternative method for computing the time-displaced Green's function based on a space-time matrix formulation of the problem. Although this technique has been successfully deployed in many-fermion simulations we won't discuss it here because of its subpar computational scaling: for a system composed of N lattice sites, fermion flavors f, and imaginary time extent M one has to invert (naively a $\mathcal{O}(x^3)$ operation) a matrix which takes up $\mathcal{O}((NMf)^2)$ memory.

A. Accuracy

In Fig. 11, we show the logarithmic maximal deviation of the time-displaced Green's function as calculated using the regular inversion scheme `inv_sum` from the exact Green's function as a function of the time-displacement τ at inverse temperature $\beta = 40$. Clearly, both non-Jacobi SVDs fail to capture the intrinsic scales sufficiently and errors much beyond floating point precision are seen. Although the QR decomposition systematically leads to equally or more accurate

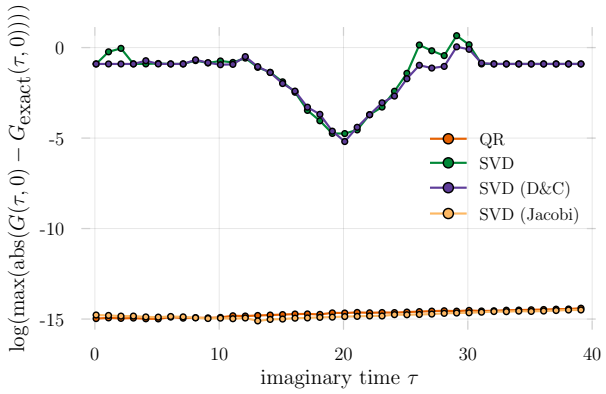

FIG. 12. **Accuracy of the time-displaced Green's function** obtained from stabilized computations using the listed matrix decompositions and the inversion scheme `inv_sum_loh`, Eq. (20), for $\beta = 40$. Shown are results for $U = 0$.

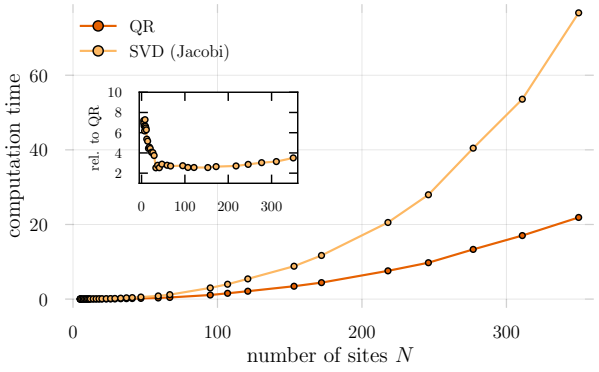

FIG. 13. **Efficiency of the time-displaced Green's function** obtained from stabilized computations using the QR decomposition in combination with the inversion scheme `inv_sum_loh`, Eq. (20) and the Jacobi SVD paired up with the regular inversion scheme `inv_sum`, Eq. (19). Measurements are taken over multiple runs at $\tau = \beta/2 = 20$. The inset show the slowdown of the Jacobi SVD relative to the QR based approach.

values for all considered imaginary times, it fails to be reliable at long times $\tau \sim \beta/2$ (recall that the Green's function is anti-periodic in τ). Only the Jacobi SVD delivers reliable results for all considered imaginary times.

When switching to the inversion scheme `inv_sum_loh`, this picture changes, as can be seen in Fig. 12. While the non-Jacobi SVDs show similar (insufficient) accuracy as when deployed in combination with `inv_sum`, using the QR decomposition leads to stable Green's function estimates up to floating point precision across the entire imaginary time axis. Similar to our findings for the equal-time Green's function, this suggests that only the Jacobi SVD and the QR decomposition, when paired with the appropriate inversion procedure, are reliable in a DQMC context.

B. Efficiency

TODO: Benchmark section where we compare Jacobi SVD + simple inversion vs QR + loh inversion.

VII. DISCUSSION

Numerical instabilities are naturally present in quantum Monte Carlo simulations of many-fermion systems. Different algorithmic schemes and matrix decomposition techniques have been proposed over time to handle the exponential spread of scales in a stable manner. However, as we have shown in this review, they can have vastly different accuracy and efficiency rendering them more or less suited for determinant quantum Monte Carlo simulations.

For the considered one-dimensional Hubbard model, we were able to compute the equal-time Green's function and its determinant to floating point precision using the QR-based UDT decomposition and the Jacobi SVD. In case of the time-displaced Green's function, the QR had to be combined with the inversion algorithm suggested by Loh *et al.* [11] while the Jacobi SVD was accurate irrespective of the inversion scheme. Conventional and divide-and-conquer based SVDs consistently failed to produce reliable results in both cases, in particular at the lowest considered temperatures, $\beta \sim 40$.

In terms of speed, we find that the QR decomposition outperforms the conventional and Jacobi SVDs by a large margin while only the D&C SVD variant has similar computational efficiency. Since the inversion scheme in the QR case involves matrix divisions this observed performance difference is not exclusively due to - but dominated by - the computational cheapness of a QR decomposition compared to a SVD.

In summary, our assessment suggests that the QR decomposition is the best choice for DQMC simulations as it is both fast and stable. However, when utilized in the computation of time-displaced Green's functions, the accuray strongly depends on the chosen inversion scheme. Among the ones considered in these notes, only the algorithm by Loh *et al.* [11] could produce reliable results. The Jacobi SVD, although computationally much more expensive, proved to be a stable alternative.

Finally, let us remark that the performance of all stabilization schemes is affected by the condition number of the time slice matrices and is therefore model (parameter) dependent. While we have not investigated this dependence in systematic detail, we nonetheless think that our major conclusions bear some universality and will hopefully serve as a useful guide,

VIII. ACKNOWLEDGEMENTS

We thank Peter Bröcker, Yoni Schattner, Snir Gazit, and Simon Trebst for useful discussions and Frederick Freyer for identifying a few typos in this manuscript.

[1] R. Blankenbecler, D. J. Scalapino, and R. L. Sugar, Monte Carlo calculations of coupled boson-fermion systems. I, Physical Review D **24**, 2278 (1981).

[2] J. E. Hirsch, Discrete Hubbard-Stratonovich transformation for fermion lattice models, Physical Review B **28**, 4059 (1983).

[3] H. F. Trotter, On the Product of Semi-Groups of Operators, Proceedings of the American Mathematical Society **10**, 545 (1959).

[4] M. Suzuki, Quantum statistical monte carlo methods and applications to spin systems, Journal of Statistical Physics **43**, 883 (1986).

[5] D. Goldberg, What every computer scientist should know about floating-point arithmetic, ACM Comput. Surv. **23**, 5 (1991).

[6] For our non-interacting model system one can alternatively simply diagonalize the Hamiltonian and calculate the Green's function exactly.

[7] Note that we do not discuss the option to calculate B^M as UD^MX. This is intentional since most real systems will involve fermion-boson interactions and the slice matrices will depend on $\phi(\tau)$.

[8] F. Assaad, *Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms*, Vol. 10 (2002) p. 99.

[9] R. R. dos Santos, Introduction to quantum Monte Carlo simulations for fermionic systems, Brazilian Journal of Physics **33**, 36 (2003).

[10] E. Y. Loh, J. E. Gubernatis, R. T. Scalettar, S. R. White, D. J. Scalapino, and R. L. Sugar, Numerical Stability and the Sign Problem in the Determinant Quantum Monte Carlo Method, International Journal of Modern Physics C **16**, 1319 (2005).

[11] E. Y. Loh, J. E. Gubernatis, R. T. Scalettar, R. L. Sugar, and S. R. White, Stable Matrix-Multiplication Algorithms for Low-Temperature Numerical Simulations of Fermions (1989) pp. 55–60.

[12] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999).

[13] Note that Fortran LAPACK functions are named according to realness and symmetries of the matrix. In Julia multiple-dispatch takes care of routing different matrix types to different *methods*. The Julia function `gesdd` works for both real and complex matrices, i.e. there is no (need for) `cgesdd`.

[14] LAPACK SVD error bounds [17] '*Thus large singular values (those near σ_1) are computed to high relative accuracy and small ones may not be.*'.

[15] For local updates on can generally avoid full calculations of Green's function determinants by exploiting locality and performing a Laplace expansion since only ratios of determinants appear in Eq. 6. In fact, in an optimal implementation the computation of the acceptance rate is $O(1)$ rather than $O(N^3)$.

[16] J. E. Hirsch, Stable Monte Carlo algorithm for fermion lattice systems at low temperatures, **38**, 12023 (1988).

[17] S. Blackford, Error Bounds for the Singular Value Decomposition, `http://www.netlib.org/lapack/lug/node96.html` (1999), [Online; accessed 16-May-2019].