Fast and stable determinant quantum Monte Carlo

Carsten Bauer[1*]

1 Institute for Theoretical Physics, University of Cologne, 50937 Cologne, Germany
* bauer@thp.uni-koeln.de

March 8, 2020

Abstract

We assess numerical stabilization methods employed in fermion many-body quantum Monte Carlo simulations. In particular, we empirically compare various matrix decomposition and inversion schemes to gain control over numerical instabilities arising in the computation of equal-time and time-displaced Green's functions within the determinant quantum Monte Carlo (DQMC) framework. Based on this comparison, we identify a procedure based on pivoted QR decompositions which is both efficient and accurate to machine precision. The Julia programming language is used for the assessment and implementations of all discussed algorithms are provided in the open-source software library `StableDQMC.jl`.

Contents

1 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 phases of matter like superconductivity or states beyond the Fermi liquid paradigm [1, 2]. 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 [3].

Fortunately, the determinant quantum Monte Carlo (DQMC) method [4–8] 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 [9, 10], 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 which, among others, has been applied to the attractive and repulsive Hubbard model [11–13], the Kane-Mele-Hubbard [14], and metallic quantum criticality, including studies of antiferromagnetic [1–3, 15], Ising-nematic [16], and deconfined quantum critical points [17] where fermionic matter fields are coupled to bosonic order parameters.

Although conceptually straightforward, care has to be taken in implementing DQMC because of inherent numerical instabilities arising from ill-conditioned matrix exponentials. Over time, many stabilization schemes [5, 8, 18–20] based on various matrix factorizations, such as singular value decomposition (SVD), modified Gram-Schmidt, and QR decomposition, have been proposed for lifting these numerical issues. It is the purpose of this manuscript to review some of these techniques and to compare them with respect to accuracy and speed. Particular emphasis is placed on concreteness and reproducibility: we provide implementations of all discussed algorithms as well as the code to recreate all plots in this manuscript in form of the software library `StableDQMC.jl`. We choose the open-source programming language Julia [21] which is aspiring [22] in the field of numerical computing and has proven [23, 24] to be capable of reaching a performance comparable to established low-level languages. Readers are invited to open issues and pull requests at the library repository to discuss, improve, and extend the list of stabilization routines. Beyond reproducibility, the software library is also intended to serve as an important abstraction layer allowing users to focus on the physical simulation instead of numerical implementation details.

Specifically, the structure of the manuscript is as follows. We start by providing a brief introduction into the DQMC method in Sec. 2. In Sec. 3 we illustrate numerical instabilities in the DQMC and discuss their origin. Following this, we demonstrate (Sec. 4) how matrix factorizations can be utilized to remedy these numerical artifacts in chains of

matrix products. In Sec. 5 we present and benchmark different schemes for stabilizing the computation of the equal-times Green's function, the fundamental building block of the DQMC method. Lastly, we turn to the calculation of time-displaced Green's functions in Sec. 6 before concluding and summarizing in Sec. 7.

2 Determinant Quantum Monte Carlo

We begin by reviewing the essentials of the determinant quantum Monte Carlo method [4–8]. Therefore, we assume a generic quantum field theory that can be split into a purely bosonic part S_B and a contribution S_F from itinerant fermions. The latter comprises both fermion kinetics, T, and boson-fermion interactions, V. A famous example is given by the Hubbard model after a decoupling of the on-site interaction $Un_{i,\uparrow}n_{i,\downarrow}$ by means of a continuous Hubbard-Stratonovich or a discrete Hirsch transformation [25][1]. The quantum statistical partition function is given by

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

The first step in DQMC is to apply the quantum-classical mapping [26] and switch from the d dimensional quantum theory above to a $D = d + 1$ dimensional classical theory. Here, the extra finite dimension of the classical theory is given by imaginary time τ and has an extent proportional to inverse temperature $\beta = 1/T$. Discretizing imaginary time into M slices, $\beta = M\Delta\tau$, and applying a Trotter-Suzuki decomposition [27,28] one obtains

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

A separation of the matrix exponential then leads to a systematic error of the order $\mathcal{O}\left(\Delta\tau^2\right)$ for the partition function,

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

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 the contribution $e^{-\Delta\tau\psi^\dagger V_\phi \psi}$ depends on the bosonic field ϕ due to potential fermion-boson coupling in V. Rewriting the trace in (3) as a determinant [8] 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

$$G = \left[\mathbb{1} + B_M B_{M-1} \cdots B_1\right]^{-1} \tag{5}$$

is the equal-time Green's function [26]. Accordingly, the Metropolis probability weight is given by

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

[1] Depending on the decomposition channel, the bosonic field ϕ represents either spin or charge fluctuations in this case.

This implies that, considering a generic, global update one needs to compute the Green's function G and it's determinant in each DQMC step[2].

Importantly, it is only under specific circumstances, such as the presence of a symmetry, that the integral kernel of the partition function can be safely interpreted as a probability weight since G_ϕ and its determinant are generally complex valued. This is the famous sign problem [29].

3 Numerical instabilities

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

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

which is free of the sign-problem [29]. We set the hopping amplitude to unity, $t = 1$[3].

As seen from Eq. (5), the building block of the equal-time Green's function is a product chain of imaginary time slice matrices. To simplify our purely numerical analysis below we assume that these slice matrices B_i are independent of imaginary time,

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

which, physically, amounts to assuming a constant bosonic field $\phi = const.$

First, we consider the non-interacting system, $U = 0$. As apparent from Fig. 1, 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 non-interacting system are readily given by

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

such that energy values are bounded by $-2t \leq \epsilon_k \leq 2t$. A single positive definite slice matrix $B = e^{-\Delta\tau T}$ therefore has a condition number of the order of $\kappa \approx e^{4|t|\Delta\tau}$ and, consequently, the product chain $B(\tau, 0)$ has $\kappa \approx e^{4|t|M\Delta\tau} = e^{4|t|\beta}$. This implies that the scales present in $B(\tau, 0)$ broaden exponentially at low temperatures $T = 1/\beta$ leading to inevitable roundoff errors due to finite machine precision which spoil the result.

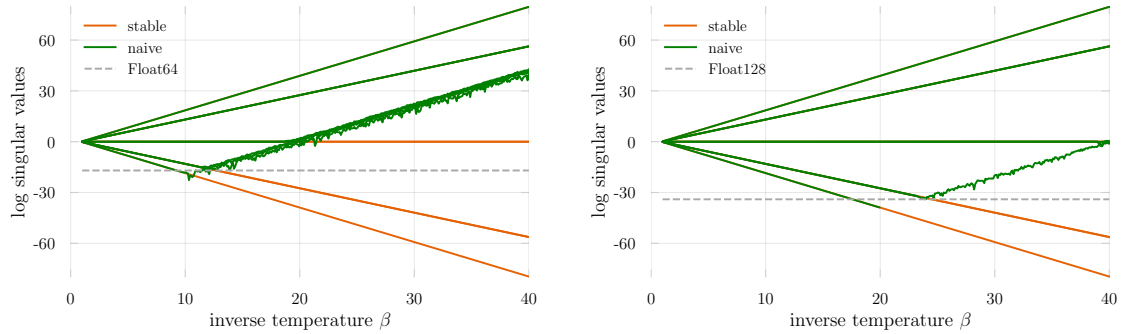
We can estimate the expected inverse temperature of this breakdown for the data type `Float64`, that is double floating-point precision according to the IEEE 754 standard [30], by solving $\kappa(\beta) \sim 10^{-17}$ for β_c. One finds $\beta_c \approx 10$ in good agreement with what is observed in Fig. 1a. Switching to the data type `Float128`[4] (quadruple precision) with $\beta_c \approx 20$ in Fig. 1b, the onset of roundoff errors is shifted to lower temperatures in accordance with expectations.

[2] For local updates one can typically avoid those explicit calculations and compute the ratio of determinants in Eq. (6) directly [2].

[3] We will consider the canonical discrete decoupling [25] in the spin channel due to Hirsch in our analysis.

[4] The datatype `Float128` is provided by the Julia package `Quadmath.jl`.

[5] We estimate the precision as $p = \log_{10}(2^{\text{fraction}})$, where fraction is the mantissa of a given binary floating point format. This gives $p \sim 16$ for `Float64` and $p \sim 34$ for `Float128`.



(a) 64-bit floating point (`Float64`) (b) 128-bit floating point (`Float128`)

Figure (1) **Numerical instabilities** due to finite machine precision (green) arising in the calculation of the slice matrix product chain $B_M B_{M-1} \cdots B_1$ for model (7). The dashed line (grey) indicates the expected floating point precision[5]. The arbitrary precision result (orange) is shown for comparison. Both considered binary floating point formats `Float64` (a) and `Float128` (b) are in compliance with the IEEE 754 standard.

4 Stabilization: matrix multiplications

4.1 Stabilization scheme

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[6]. 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[7].

How can we get a handle on the numerical instabilities in a floating point precision computation? The strategy is to keep the broadly different scales in the matrix exponentials separated throughout the computation (as much as possible) and only mix them in the final step, if necessary. A useful tool for extracting the scale information is a matrix decomposition,

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

Here, U and X are matrices of order unity and D is a real diagonal matrix hosting the exponentially spread scales of B. We will refer to the values in D as singular values independent of the particular decomposition. Using Eq. (10), we can stabilize the matrix multiplication of slice matrices B_1 and B_2 in Eq. (8) as follows, (`fact_mult` in `StableDQMC.jl`)

$$\begin{aligned} B_2 B_1 &= \underbrace{U_2 D_2 X_2}_{B_2} \underbrace{U_1 D_1 X_1}_{B_1} \\ &= U_2 \underbrace{\left(D_2((X_2 U_1) D_1)\right)}_{U' D' X'} X_1) \\ &= U_r D_r X_r. \end{aligned} \tag{11}$$

Here, $U_r = U_2 U'$, $D_r = D'$, $X_r = X' X_1$, and $U'D'X'$ indicates an intermediate matrix decomposition. If we follow this scheme, in which parentheses indicate the order of

[6] Technically, `BigFloat` has a finite, arbitrarily high precision.

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

operations, largely different scales present in the diagonal matrices won't be additively mixed throughout the computation. Specifically, note that the multiplication of the well-conditioned, combined, unit-scale matrix $U = X_2 U_1$ with D_1 and D_2 does preserve the scale information: the diagonal matrices merely rescale the columns and rows of U,

$$D_2 U D_1 = \begin{bmatrix} \mathrm{S} & & & \\ & \mathrm{S} & & \\ & & \mathrm{s} & \\ & & & {}_{\mathrm{s}} \end{bmatrix} \underbrace{\begin{bmatrix} \mathrm{s} & \mathrm{s} & \mathrm{s} & \mathrm{s} \\ \mathrm{s} & \mathrm{s} & \mathrm{s} & \mathrm{s} \\ \mathrm{s} & \mathrm{s} & \mathrm{s} & \mathrm{s} \\ \mathrm{s} & \mathrm{s} & \mathrm{s} & \mathrm{s} \end{bmatrix}}_{\text{unit scale}} \begin{bmatrix} \mathrm{S} & & & \\ & \mathrm{S} & & \\ & & \mathrm{s} & \\ & & & {}_{\mathrm{s}} \end{bmatrix} \tag{12}$$

$$= \begin{bmatrix} \mathrm{S} & & & \\ & \mathrm{S} & & \\ & & \mathrm{s} & \\ & & & {}_{\mathrm{s}} \end{bmatrix} \begin{bmatrix} \mathrm{sS} & \mathrm{sS} & \mathrm{s}^2 & \mathrm{s}_{\mathrm{s}} \\ \mathrm{sS} & \mathrm{sS} & \mathrm{s}^2 & \mathrm{s}_{\mathrm{s}} \\ \mathrm{sS} & \mathrm{sS} & \mathrm{s}^2 & \mathrm{s}_{\mathrm{s}} \\ \mathrm{sS} & \mathrm{sS} & \mathrm{s}^2 & \mathrm{s}_{\mathrm{s}} \end{bmatrix} \tag{13}$$

$$= \begin{bmatrix} \mathrm{S}^2\mathrm{s} & \mathrm{SSs} & \mathrm{Ss}^2 & \mathrm{Ss}_{\mathrm{s}} \\ \mathrm{SSs} & \mathrm{S}^2\mathrm{s} & \mathrm{Ss}^2 & \mathrm{Ss}_{\mathrm{s}} \\ \mathrm{Ss}^2 & \mathrm{Ss}^2 & \mathrm{s}^3 & \mathrm{s}^2{}_{\mathrm{s}} \\ \mathrm{Ss}_{\mathrm{s}} & \mathrm{Ss}_{\mathrm{s}} & \mathrm{s}^2{}_{\mathrm{s}} & \mathrm{s}_{\mathrm{s}}{}^2 \end{bmatrix}. \tag{14}$$

Repeating the procedure (11), we obtain a numerically accurate UDX decomposition of the full slice matrix product chain $B(\tau, 0)$, which preserves the scale information as indicated in Fig. 1.[8] We note in passing that in practice it is often unnecessary to stabilize every individual matrix product. Instead one typically performs a mixture of naive and stabilized products for the sake of speed while still retaining numerical accuracy [8].

4.2 Matrix decompositions

There are a various matrix decompositions that one could employ to obtain the factorization $B = UDX$, Eq (10). In the following we will consider the two most popular choices in DQMC codes [7, 8, 18].

4.2.1 SVD ($UDV^\dagger$)

The singular value decomposition (SVD) is given by

$$B = USV^\dagger, \tag{15}$$

where U and $V^\dagger$ are unitary and S is real and diagonal.

For computing the SVD of a matrix of regular floating point precision (`Matrix{Float64}`), Julia utilizes the heavily optimized routines provided by LAPACK [31]. Concretely, there exist three different implementations of SVD algorithms:[9]

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

[8] Note that we do not discuss the faster way 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 $\phi(\tau)$.

[9] 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`.

To simplify the manual access to these algorithms we export convenience wrappers of the same name in `StableDQMC.jl`. We will compare all three variants below and benchmark them against an arbitrary precision computation using `BigFloat`. Since LAPACK doesn't support special number types, we will utilize the native-Julia SVD implementation provided by GenericSVD.jl in this case.

4.2.2 QR (UDT)

A QR decomposition reads

$$B = QR = UDT, \tag{16}$$

where Q is unitary, R is upper triangular, and we have split R into a diagonal part D and an upper triangular part T in the second step. Specifically, $U = Q$ is unitary, $D = \mathrm{diag}(R)$ is real and diagonal, 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`[10]. A factorization into UDT form is provided by analogous functions `udt` and `udt!` in `StableDQMC.jl`.

5 Stabilization: equal-time Green's function

Similar to the considerations in Sec. 3, a naive computation of the Green's function according to Eq. (5) is potentially unstable because of numerical roundoff errors due to finite machine precision. In particular, adding the identity to the ill-conditioned slice matrix chain $B(\tau, 0)$ will generally wash out small singular values and will lead to a non-invertible result such that the subsequent inversion in Eq. (5) is ill-defined. This clearly prohibits a safe calculation of the equal-time Green's function and asks for numerical stabilization techniques.

5.1 Inversion schemes

As for the matrix products in Eq. (11), the strategy will be to keep exponentially spread scales as separated as possible. A straightforward scheme [7,8] (`inv_one_plus`) to add the unit matrix and perform the inversion of $\mathbb{1} + B(\tau, 0)$ in a stabilized manner is given by

$$\begin{aligned} G &= [\mathbb{1} + UDX]^{-1} \\ &= [U \underbrace{(U^\dagger X^{-1} + D)}_{udx} X]^{-1} \\ &= [(Uu)d(xX)]^{-1} \\ &= U_r D_r X_r, \end{aligned} \tag{17}$$

where $U_r = (xX)^{-1}$, $D_r = d^{-1}$, and $X_r = (Uu)^{-1}$. Here, the intermediate addition (parentheses in the second line of (17)) of unit scales and singular values is separated from the unitary rotations such that $U^\dagger X^{-1}$ only acts as a clean cutoff,

$$U^\dagger X^{-1} + D = \begin{bmatrix} \mathrm{s} & \mathrm{s} & \mathrm{s} & \mathrm{s} \\ \mathrm{s} & \mathrm{s} & \mathrm{s} & \mathrm{s} \\ \mathrm{s} & \mathrm{s} & \mathrm{s} & \mathrm{s} \\ \mathrm{s} & \mathrm{s} & \mathrm{s} & \mathrm{s} \end{bmatrix} + \begin{bmatrix} \mathrm{S} & & & \\ & \mathrm{S} & & \\ & & \mathrm{s} & \\ & & & {}_{\mathrm{s}} \end{bmatrix} = \begin{bmatrix} \mathrm{S} & \mathrm{s} & \mathrm{s} & \mathrm{s} \\ \mathrm{s} & \mathrm{S} & \mathrm{s} & \mathrm{s} \\ \mathrm{s} & \mathrm{s} & \mathrm{s} & \mathrm{s} \\ \mathrm{s} & \mathrm{s} & \mathrm{s} & \mathrm{s} \end{bmatrix}. \tag{18}$$

[10] We consider the pivoted QR throughout our analysis.

However, as it turns out, this scheme will nonetheless fail to give accurate results for the equal-time Green's function when combined with certain matrix decompositions, as we will demonstrate below. For this reason, we consider another stabilization procedure put forward by Loh *et al.* [5,18] (`inv_one_plus_loh`), in which one initially separates the scales of the diagonal matrix D into two factors $D_p = \max(D, 1)$ and $D_m = \min(D, 1)$,

$$D_p = \begin{bmatrix} \mathrm{S} & & & \\ & \mathrm{S} & & \\ & & \mathrm{s} & \\ & & & \mathrm{s} \end{bmatrix}, \quad D_m = \begin{bmatrix} \mathrm{s} & & & \\ & \mathrm{s} & & \\ & & \mathrm{s} & \\ & & & \mathrm{s} \end{bmatrix}, \tag{19}$$

and perform two intermediate decompositions,

$$\begin{aligned} G &= [\mathbb{1} + UDX]^{-1} \\ &= [\mathbb{1} + UD_mD_pX]^{-1} \\ &= [(X^{-1}D_p^{-1} + UD_m)D_pX]^{-1} \\ &= X^{-1}\underbrace{[D_p^{-1}(\underbrace{X^{-1}D_p^{-1} + UD_m}_{udx})^{-1}]}_{udx} \\ &= U_rD_rX_r, \end{aligned} \tag{20}$$

where $U_r = X^{-1}u$, $D_r = d$, and $X_r = x$.

5.2 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)

5.2.1 Accuracy

Before benchmarking the efficiency of an algorithm, it is crucial to check it's correctness first. Fig. 2 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 ($\beta \gtrsim 25$).[11]

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. 17, are shown in Fig. 3a. 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. 3b, switching to the more careful procedure `inv_one_plus_loh`, Eq. 20, 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.

[11] The fact that small scales are lost while large scales remain relatively accurate can be understood from LAPACK's SVD errorbounds [32].

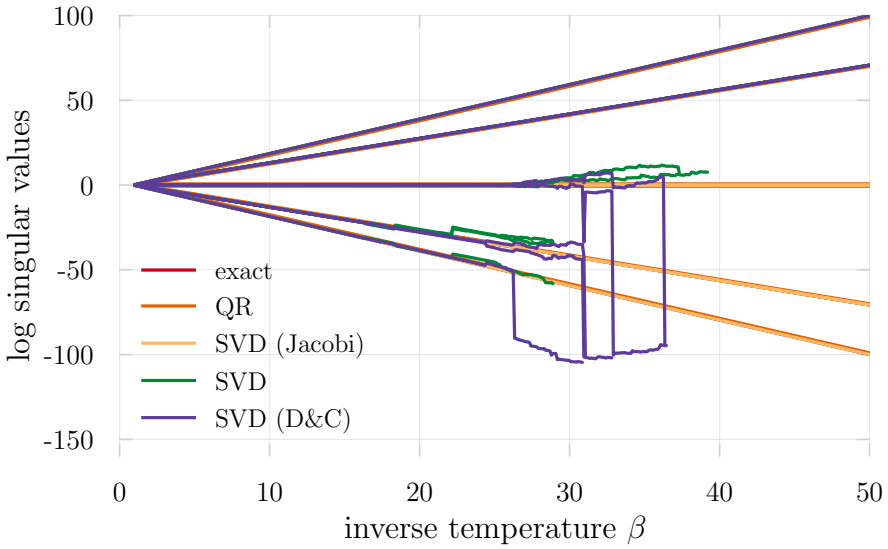
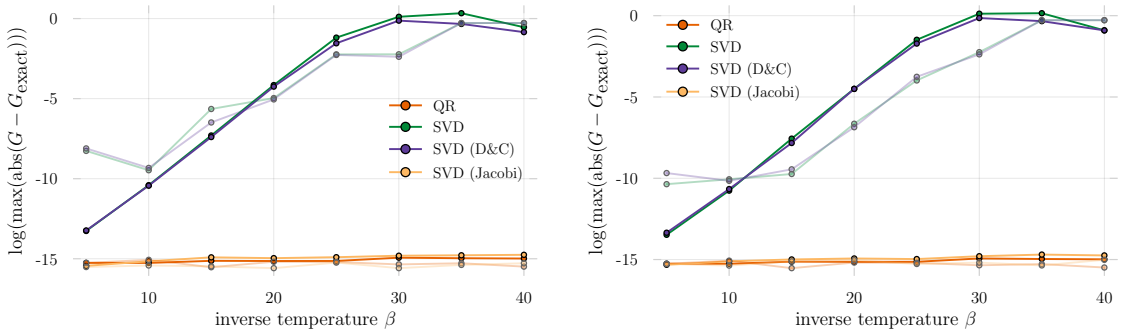
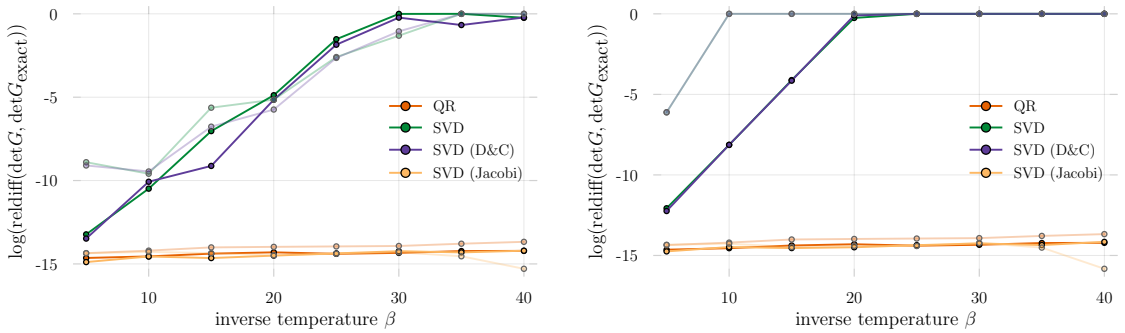


Figure (2) **Comparison of matrix decompositions** to heal the numerical instabilities in the calculation of the slice matrix product chain $B_M B_{M-1} \cdots B_1$ for model (7). The QR and Jacobi SVD singular values seem to lie on top of the exact ones whereas regular SVD and divide-and-conquer SVD show large deviations at low temperatures $\beta \gtrsim 25$ ($\Delta\tau = 0.1$).



(a) `inv_one_plus`, Eq. (17) (b) `inv_one_plus_loh`, Eq. 20

Figure (3) **Accuracy of the Green's function** obtained from stabilized computations using the listed matrix decompositions and the inversion schemes. Shown are results for $U = 0$ (solid) and $U = 1$ (alpha transparent).



(a) `inv_one_plus`, Eq. 17. (b) `inv_one_plus_loh`, Eq. 20

Figure (4) **Accuracy of the determinant** of the equal-time Green's function obtained from stabilized computations using the listed matrix decompositions and the inversion schemes. Shown are results for $U = 0$ (solid) and $U = 1$ (alpha transparent).

In Figs. 4a, 4b we show the logarithm of the relative error of the Green's function determinant, relevant in the Metropolis acceptance[12], obtained for all combinations of matrix decompositions and inversion schemes. Both the QR decomposition 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.

5.2.2 Efficiency

Independent of the deployed inversion scheme, matrix decompositions account for most of the computational cost in the Green's function calculation. Fig. 5 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 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. 3a and 3b, 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.

6 Stabilization: time-displaced Green's function

TODO: Motivate TDGF. Why interesting? It is not need in the plain DQMC.

[12] 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 $O(N^3)$.

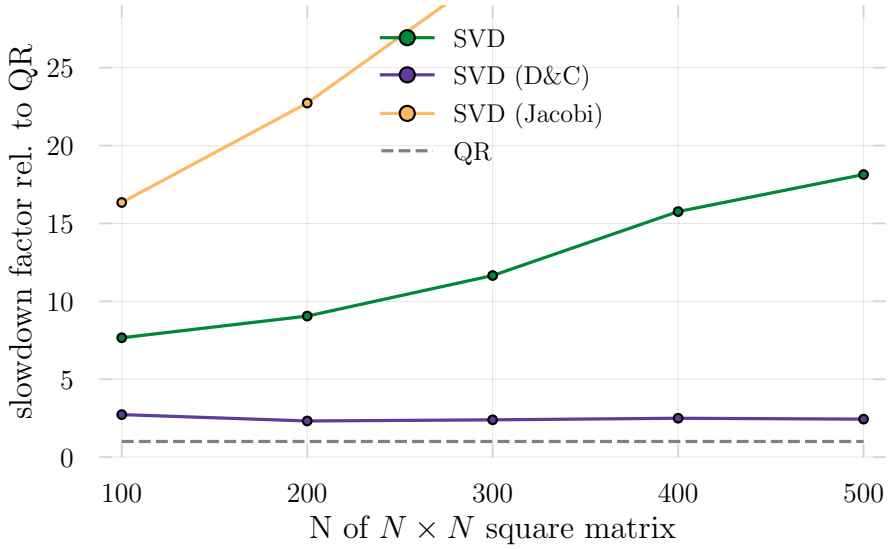
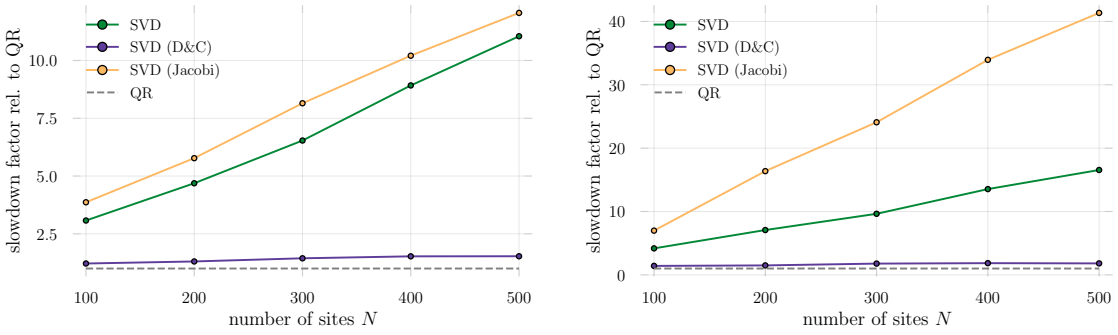


Figure (5) **Efficiency of different matrix decompositions**. Shown are the slowdown factors of single SVDs relative to a QR decomposition of a complex matrix of size $N \times N$.



(a) `inv_one_plus`, Eq. (17). (b) `inv_one_plus_loh`, Eq. (20).

Figure (6) **Efficiency of the stabilized Green's function calculation** using the listed matrix decompositions and the inversion schemes. Shown are results for $U = 0$.

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

$$G(\tau) = \langle c_i c_j^\dagger \rangle_{\phi_l} = (1 + B_{l-1} \dots B_1 B_M \dots B_l)^{-1} . \tag{21}$$

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 [7, 8]

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

where T represents time ordering.

More explicitly this reads

$$G(\tau_1, \tau_2) = \begin{cases} B_{l_1} \cdots B_{l_2+1} G_{l_2+1}, & \tau_1 > \tau_2, \\ -(1 - G_{l_1+1}) (B_{l_2} \cdots B_{l_1+1})^{-1}, & \tau_2 > \tau_1. \end{cases} \tag{22}$$

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. 22 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

$$G(\tau_1, \tau_2) = (U_L D_L X_L + U_R D_R X_R)^{-1}. \tag{23}$$

6.1 Inversion schemes

Similar to Sec. 4, 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. 17 (`inv_sum`),

$$\begin{aligned} G(\tau_1, \tau_2) &= [U_L D_L X_L + U_R D_R X_R]^{-1} \\ &= [U_L \underbrace{(D_L X_L X_R^{-1} + U_L^\dagger U_R D_R)}_{udx} X_R]^{-1} \\ &= [(U_L u) d^{-1} (x X_R)]^{-1} \\ &= U_r D_r X_r, \end{aligned} \tag{24}$$

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

A different scheme, analogous to Eq. 20, where we split the scales in D, is as follows (`inv_sum_loh`), [5]

$$\begin{aligned} G(\tau_1, \tau_2) &= [U_L D_L X_L + U_R D_R X_R]^{-1} \\ &= [U_L D_{Lm} D_{Lp} X_L + U_R D_{Rm} D_{Rp} X_R]^{-1} \\ &= \left[U_L D_{Lp} \underbrace{\left(\frac{D_{Lm}}{D_{Rp}} X_L X_R^{-1} + U_L^\dagger U_R \frac{D_{Rm}}{D_{Lp}} \right)}_{udx} X_R D_{Rp} \right]^{-1} \\ &= X_R^{-1} \underbrace{\frac{1}{D_{Rp}} [udx]^{-1} \frac{1}{D_{Lp}}}_{udx} U_L^\dagger \\ &= U_r D_r X_r, \end{aligned} \tag{25}$$

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

We note in passing that [33] 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. [mention Assaad's 2x2 matrix approach for getting both equal and time-displaced Green's functions]

6.2 Benchmarks

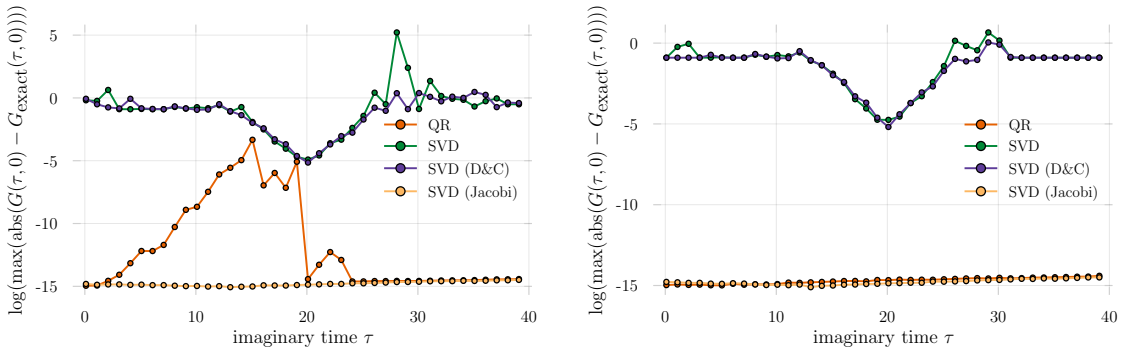
6.2.1 Accuracy

In Fig. 7a, 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 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`, the situation changes, as can be seen in Fig. 7b. 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.

6.2.2 Efficiency

Finally, we compare the computational efficiency of the Jacobi SVD combined with the regular inversion and the QR decomposition paired with `inv_sum_loh`. As shown in Fig. 8, we find that the latter is consistently faster for all considered system sizes. In relative terms, the SVD based approach is at least a factor of two slower and seems to display



(a) `inv_sum`, Eq. (24). (b) `inv_sum_loh`, Eq. (25).

Figure (7) **Accuracy of the time-displaced Green's function** obtained from stabilized computations using the listed matrix decompositions and the inversion schemes for $\beta = 40$.

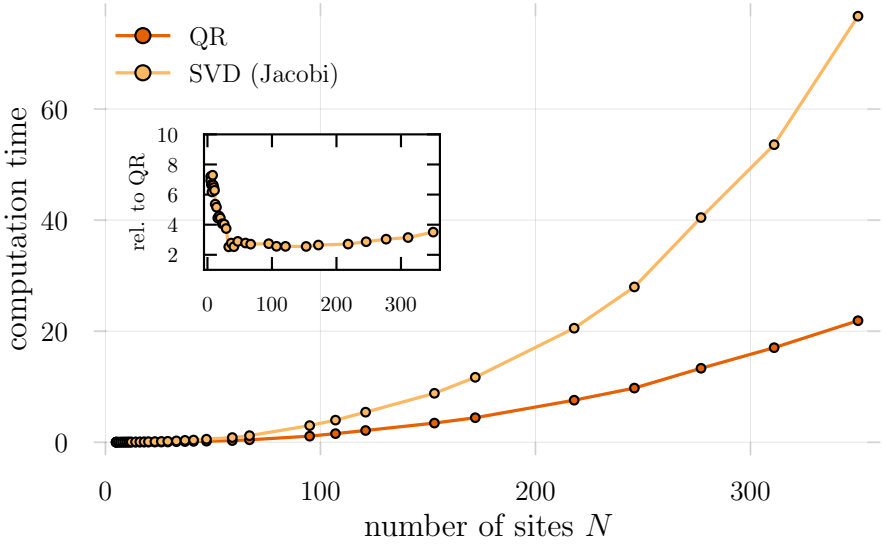


Figure (8) **Efficiency of the time-displaced Green's function** obtained from stabilized computations using the QR decomposition in combination with the inversion scheme `inv_sum_loh`, Eq. (25) and the Jacobi SVD paired up with the regular inversion scheme `inv_sum`, Eq. (24). Measurements are taken over multiple runs at $\tau = \beta/2 = 20$. The inset show the slowdown of the Jacobi SVD relative to the QR based approach.

inferior scaling with the chain length N. This indicates that in a DQMC context numerical stabilization via QR decompositions should be preferred despite the higher complexity of the inversion scheme.

7 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 these notes, 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 [18] 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 [18] could produce reliable results. The Jacobi SVD, although computationally more expensive, proved to be a stable alternative. We note that interested readers may find a more analytical comparison of the performance of QR and Jacobi SVD for DQMC in Ref. [19].

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.

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.

Funding information Mention CRC 183 funding here?

A Inversion schemes for a slice matrix function stack

In practical implementations of DQMC one typically stores intermediate decomposed slice matrix products $B(\tau_1, \tau_2)$ in a stack for reuse in future Green's function calculations [7, 8, 34]. In this case, the inversion procedure in Eq. (17) needs to be supplemented by a scheme to combine two elements U_L, D_L, X_L and U_R, D_R, X_R from the stack. Below we describe the latter for both matrix decompositions considered in the main text[13].

A.1 QR/UDT

(`StableDQMC.jl`: `inv_one_plus(::UDT, ::UDT)`)

$$\begin{aligned} G &= \left[\mathbb{1} + U_L D_L T_L \left(U_R D_R T_R\right)^\dagger\right]^{-1} \\ &= \left[\mathbb{1} + U_L \underbrace{\left(D_L \left(T_L T_R^\dagger\right) D_R\right)}_{udt} U_R^\dagger\right]^{-1} \\ &= \left[\mathbb{1} + UDT\right]^{-1}, \end{aligned} \tag{26}$$

with $U = U_L u$, $D = d$, and $T = t U_R^\dagger$. This UDT factorization may now be substituted into Eq. (17).

[13] In `StableDQMC.jl`, Julia's multiple dispatch will automatically select the correct method based on the number of provided UDT factorizations.

A.2 SVD

(`StableDQMC.jl`: `inv_one_plus(::SVD, ::SVD)`)

$$\begin{aligned} G &= \left[\mathbb{1} + U_L D_L V_L^\dagger U_R D_R V_R^\dagger\right]^{-1} \\ &= \left[\mathbb{1} + U_L \underbrace{\left(D_L \left(V_L^\dagger U_R\right) D_R\right)}_{udv^\dagger} V_R^\dagger\right]^{-1} \\ &= \left[\mathbb{1} + UDV^\dagger\right]^{-1}, \end{aligned} \tag{27}$$

with $U = U_L u$, $D = d$, and $V = v V_R$. This SVD factorization may now be substituted into Eq. (17).

References

[1] C. Bauer, Y. Schattner, S. Trebst and E. Berg, *Hierarchy of energy scales in an O(3) symmetric antiferromagnetic quantum critical metal: a Monte Carlo study* (3), 1 (2020), 2001.00586.

[2] Y. Schattner, M. H. Gerlach, S. Trebst and E. Berg, *Competing Orders in a Nearly Antiferromagnetic Metal*, Phys. Rev. Lett. **117**(9), 097002 (2016), doi:10.1103/PhysRevLett.117.097002.

[3] E. Berg, S. Lederer, Y. Schattner and S. Trebst, *Monte Carlo Studies of Quantum Critical Metals*, Annual Review of Condensed Matter Physics **10**(1), 63 (2019), doi:10.1146/annurev-conmatphys-031218-013339.

[4] R. Blankenbecler, D. J. Scalapino and R. L. Sugar, *Monte Carlo calculations of coupled boson-fermion systems. I*, Physical Review D **24**(8), 2278 (1981), doi:10.1103/PhysRevD.24.2278.

[5] 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**(08), 1319 (2005), doi:10.1142/S0129183105007911.

[6] D. J. Scalapino, S. R. White and S. Zhang, *Insulator, metal, or superconductor: The criteria*, Phys. Rev. B **47**(13), 7995 (1993), doi:10.1103/PhysRevB.47.7995.

[7] R. R. dos Santos, *Introduction to quantum Monte Carlo simulations for fermionic systems*, Brazilian Journal of Physics **33**(1), 36 (2003), doi:10.1590/S0103-97332003000100003.

[8] F. Assaad, *Quantum Monte Carlo Methods on Lattices: The Determinantal Approach*, vol. 10, ISBN 3000090576 (2002).

[9] E. Y. Loh, J. E. Gubernatis, R. T. Scalettar, S. R. White, D. J. Scalapino and R. L. Sugar, *Sign problem in the numerical simulation of many-electron systems*, Physical Review B **41**(13), 9301 (1990), doi:10.1103/PhysRevB.41.9301.

[10] M. Troyer and U.-J. Wiese, *Computational complexity and fundamental limitations to fermionic quantum Monte Carlo simulations*, Physical review letters **94**(17), 170201 (2005), doi:10.1103/PhysRevLett.94.170201, 0408370.

[11] J. E. Hirsch, *Two-dimensional Hubbard model: Numerical simulation study*, Physical Review B **31**(7), 4403 (1985), doi:10.1103/PhysRevB.31.4403.

[12] S. White, D. Scalapino, R. Sugar, E. Loh, J. Gubernatis and R. Scalettar, *Numerical study of the two-dimensional Hubbard model*, Physical Review B **40**(1), 506 (1989), doi:10.1103/PhysRevB.40.506.

[13] A. Moreo and D. J. Scalapino, *Two-dimensional negative-U Hubbard model*, Physical Review Letters **66**(7), 946 (1991), doi:10.1103/PhysRevLett.66.946.

[14] M. Hohenadler, Z. Y. Meng, T. C. Lang, S. Wessel, A. Muramatsu and F. F. Assaad, *Quantum phase transitions in the Kane-Mele-Hubbard model*, Physical Review B - Condensed Matter and Materials Physics **85**(11), 1 (2012), doi:10.1103/PhysRevB.85.115132, 1111.3949.

[15] M. H. Gerlach, Y. Schattner, E. Berg and S. Trebst, *Quantum critical properties of a metallic spin-density-wave transition*, Phys. Rev. B **95**(3), 035124 (2017), doi:10.1103/PhysRevB.95.035124.

[16] Y. Schattner, S. Lederer, S. A. Kivelson and E. Berg, *Ising Nematic Quantum Critical Point in a Metal: A Monte Carlo Study*, Phys. Rev. X **6**(3), 031028 (2016), doi:10.1103/PhysRevX.6.031028.

[17] S. Gazit, M. Randeria and A. Vishwanath, *Emergent Dirac fermions and broken symmetries in confined and deconfined phases of Z 2 gauge theories*, Nature Physics **13**(5), 484 (2017), doi:10.1038/nphys4028.

[18] 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*, pp. 55–60, doi:10.1007/978-1-4613-0565-1_8 (1989).

[19] Z. Bai, C. Lee, R. C. Li and S. Xu, *Stable solutions of linear systems involving long chain of matrix multiplications*, Linear Algebra and Its Applications **435**(3), 659 (2011), doi:10.1016/j.laa.2010.06.023.

[20] S. Sorella, S. Baroni, R. Car and M. Parrinello, *A novel technique for the simulation of interacting fermion systems*, Epl **8**(7), 663 (1989), doi:10.1209/0295-5075/8/7/014.

[21] J. Bezanson, A. Edelman, S. Karpinski and V. B. Shah, *Julia: A Fresh Approach to Numerical Computing*, SIAM Review **59**(1), 65 (2017), doi:10.1137/141000671.

[22] J. M. Perkel, *Julia: come for the syntax, stay for the speed*, Nature **572**, 141 (2019), doi:10.1038/d41586-019-02310-3, [Online; accessed 4-March-2020].

[23] C. Rackauckas, *A Comparison Between Differential Equation Solver Suites In MATLAB, R, Julia, Python, C, Mathematica, Maple, and Fortran*, https://www.stochasticlifestyle.com/comparison-differential-equation-solver-suites-matlab-r-julia-python-c-fortran/, [Online; accessed 4-March-2020].

[24] X.-Z. Luo, J.-G. Liu, P. Zhang and L. Wang, *Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design* (2019), 1912.10877.

[25] J. E. Hirsch, *Discrete Hubbard-Stratonovich transformation for fermion lattice models*, Physical Review B **28**(7), 4059 (1983), doi:10.1103/PhysRevB.28.4059.

[26] S. Sachdev, *Quantum Phase Transitions*, In *Quantum Phase Transitions*. Cambridge University Press, ISBN 9780521514682 (2011).

[27] H. F. Trotter, *On the Product of Semi-Groups of Operators*, Proceedings of the American Mathematical Society **10**(4), 545 (1959), doi:https://doi.org/10.2307/2033649.

[28] M. Suzuki, *Quantum statistical monte carlo methods and applications to spin systems*, Journal of Statistical Physics **43**(5-6), 883 (1986), doi:10.1007/BF02628318.

[29] Z.-X. Li and H. Yao, *Sign-Problem-Free Fermionic Quantum Monte Carlo: Developments and Applications*, Annual Review of Condensed Matter Physics **10**(1), 337 (2019), doi:10.1146/annurev-conmatphys-033117-054307.

[30] D. Goldberg, *What every computer scientist should know about floating-point arithmetic*, ACM Comput. Surv. **23**(1), 5 (1991), doi:10.1145/103162.103163.

[31] 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*, Society for Industrial and Applied Mathematics, Philadelphia, PA, third edn., ISBN 0-89871-447-8 (paperback) (1999).

[32] S. Blackford, *Error Bounds for the Singular Value Decomposition*, `http://www.netlib.org/lapack/lug/node96.html`, [Online; accessed 4-March-2020] (1999).

[33] J. E. Hirsch, *Stable Monte Carlo algorithm for fermion lattice systems at low temperatures* **38**(16), 12023 (1988), doi:http://dx.doi.org/10.4236/ojo.2014.48035.

[34] P. Broecker and S. Trebst, *Numerical stabilization of entanglement computation in auxiliary-field quantum Monte Carlo simulations of interacting many-fermion systems*, Physical Review E **94**(6), 063306 (2016), doi:10.1103/PhysRevE.94.063306.