Notes on numerical stabilization in determinant quantum Monte Carlo

Carsten Bauer[1]

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

(Dated: May 27, 2019)

In these notes we will empirically compare different matrix decompositions for use in numerical stabilization in many-body quantum Monte Carlo simulation, in particular when calculating equal time Green's functions. We will both benchmark their speed and accuracy. We will further show that special care has to be taken when calculating time-displaced Green's functions and will review a well-known but somewhat hard to find method for stabilizing the necessary inversion in this case. We will focus on and use the Julia programming language for all calculations. An implementation of all discussed methods is available in the open-source software library `StableDQMC.jl`.

I. THE ISSUE

A. Determinant quantum Monte Carlo in a nutshell

For the description of the DQMC algorithm,[1] we consider a quantum field theory that can be split into a purely bosonic part S_B and a part S_F, which 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 is to switch from the d dimensional quantum theory to a $D = d+1$ dimensional classical theory as in the path integral framework. The first step is to discretize, $\beta = M\Delta\tau$, and Trotter decompose the imaginary time τ,

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

The exponential is then separated and the appearing commutator ignored which results in a systematic error of the order $\mathcal{O}\left(\Delta\tau^2\right)$,

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

Note that the potential part $e^{-\Delta\tau\psi^\dagger V_\phi \psi}$ in the time-slice propagators $B_l = e^{-\frac{\Delta\tau}{2}\psi^\dagger T \psi} e^{-\Delta\tau\psi^\dagger V_\phi\psi} e^{-\frac{\Delta\tau}{2}\psi^\dagger T\psi}$ depends on the boson ϕ due to the present 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

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

is the equal-time Green's function.

Only under specific circumstances, such as the presence of a symmetry, can the integral kernel be safely be interpreted as a probability weight, as G_ϕ and its determinant are generally complex valued (*sign problem*).

B. Numerical instabilities - Model system

We consider the following simple non-interacting model system in one dimension,

$$H = -t \sum_{\langle i,j\rangle} c_i^\dagger c_j + \mu \sum_i n_i, \tag{6}$$

where we set the hopping amplitude to $t = -1$ and the chemical potential to $\mu = -0.1$. Generally, we are interested in the equal time Green's function G, Eq. (5), as an observable and it's determinant, which enters into the probability weight in Eq. (4). Let's make the latter point a bit more precise. In a Metropolis scheme,

$$p = \min\left\{1, e^{-\Delta S_\phi} \frac{\det G}{\det G'}\right\}, \tag{7}$$

it is actually the ratio of determinants which determines the acceptance or rejection of a proposed (primed G) Markov walker step.

We will showcase the issue of numerical instabilities arising in the computation of G by discussing the accuracy of the calculation of the building block of G, the slice matrix product chain

$$B_M B_{M-1} \cdots B_1 = \underbrace{BB\cdots B}_{M \text{ factors}}. \tag{8}$$

Here the second equality stems from the fact that our model, Eq. 6, is non-interacting and the slice matrices B_l

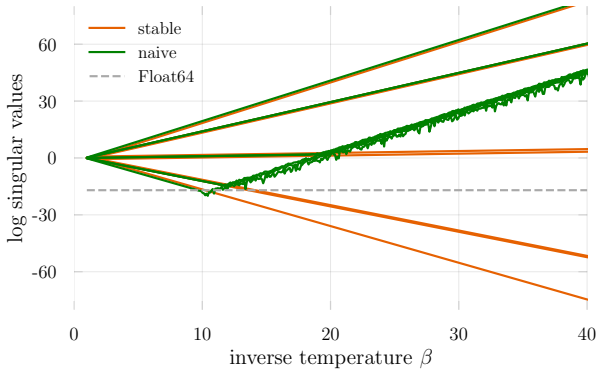

FIG. 1. **Numerical instabilities** (green) due to finite machine precision (`Float64`) in the calculation of the slice matrix product chain $B_M B_{M-1} \cdots B_1$ for model (6).

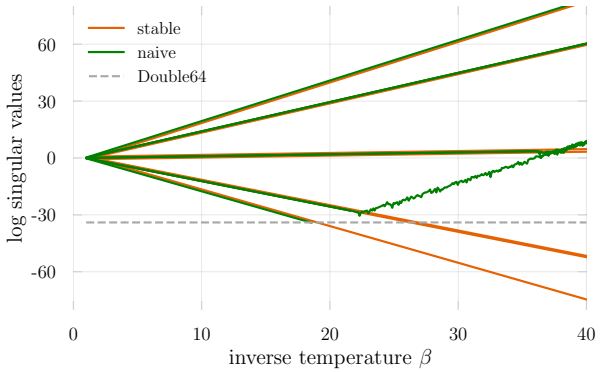

FIG. 2. **Numerical instabilities** due to finite machine precision (`Double64`) in the calculation of the slice matrix product chain $B_M B_{M-1} \cdots B_1$ for model (6).

do neither depend on any Hubbard-Stratonovich boson field nor on imaginary time.

In Fig. 1 we see that a naive computation of Eq. 8 is doomed to fail for $M \geq M_c \approx 100$. Leaving a discussion of how to stabilize the product for the next section, let us understand the origin of this instability. The eigenvalues of the system are given by

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

Neglecting the contribution by the chemical potential for simplicity, the energy values are bounded by $-2t \leq \epsilon_k \leq 2t$. The positive definite slice matrix $B = e^{-\Delta\tau T}$ has a condition number of about $\kappa \approx e^{4|t|\Delta\tau}$ and the product chain, Eq. 8, has $\kappa \approx e^{4|t|M\Delta\tau} = e^{4|t|\beta}$. We therefore see that the condition number diverges at low temperatures $T = 1/\beta = 1/(M\Delta\tau)$ where M is large. In this case roundoff errors due to finite machine precision will spoil the result. We can estimate the breakdown of the computation for the data type `Float64`, that is double floating-point precision,[3] by solving $\kappa(M) \sim 10^{-17}$ for M_c. We find $M_c \approx 100$ 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.

II. STABILIZATION SCHEMES

One way to solve all problems is to switch to arbitrary precision numerics. In Julia the latter is provided through the `BigFloat` data type. However, this comes at the expense of (very) slow performance since, among other things, `BigFloat` lacks hardware support. Arbitrary precision numerics is still a valuable tool and we will use it to benchmark the accuracy of stabilization methods below.

How can we remedy the numerical issue outlined above in a floating point precision computation? The idea is to keep different scales separated throughout the computation as long as possible and only mix them in the final step, if necessary. A way to realize this is by deploying matrix decompositions, which we write generally as

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

Here, U and X are matrices containing scales of the order of unity and D is a real diagonal matrix having all the scales of B separated on the diagonal. We will refer to the values in D as singular values for all particular decompositions.

Instead of calculating the matrix product B_2B_1 directly, we safely compute (`fact_mult` in `StableDQMC.jl`)

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

with $U_r = U_2U'$, $D_r = D'$, and $X_r = X'X_1$. Here, brackets indicate the order of operations. If we follow this scheme, largely different scales contained in the diagonal matrices won't be mixed throughout the computation. Repeating this procedure for the full slice matrix product chain, Eq. (8), we obtain an accurate UDX decomposition of the result.[4] TODO: Mention safe mult

Using this scheme, we have to be careful to keep different scales separated in the equal-time Green's function calculation, Eq. 5, as well. A straightforward procedure (`inv_one_plus` in `StableDQMC.jl`) to add the unit matrix and perform the inversion in a stabilized manner is given by[5,6]

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

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

Another prescription for a stabilized inversion (`inv_one_plus_loh` in `StableDQMC.jl`), 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[7,8]

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

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.

A. 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[9] and Julia defaults to those algorithms for regular floating point types. Concretely, there are three SVD functions[10] 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.

B. QR (UDT)

A QR decomposition reads

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

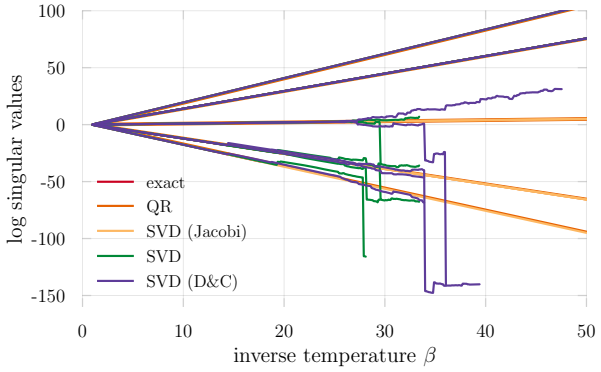

FIG. 3. **Comparison of matrix decompositions** to heal the numerical instabilities in the calculation of the slice matrix product chain $B_MB_{M-1}\cdots B_1$ for model (6). The QR and Jacobi SVD singular values seem to lie on top of the exact ones whereas regular SVD and divide-and-conquer SVD show large deviations at low temperatures $\beta \gtrsim 25$ ($\Delta\tau = 0.1$).

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`. The splitting into UDT form is provided by `udt` and `udt!` in in `StableDQMC.jl`.

III. BENCHMARKS

In the following we want to assess how the mentioned matrix decompositions perform in stabilized computations of the slice matrix product chain $B_M\cdots B_1$, the Green's function G, and it's determinant $\det G$, both with respect to accuracy and speed, for our model system in Eq. 6.

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 Eq. 8 stabilized with different matrix decompositions as a function of it's length M. 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. 12, are shown in Fig. 4.

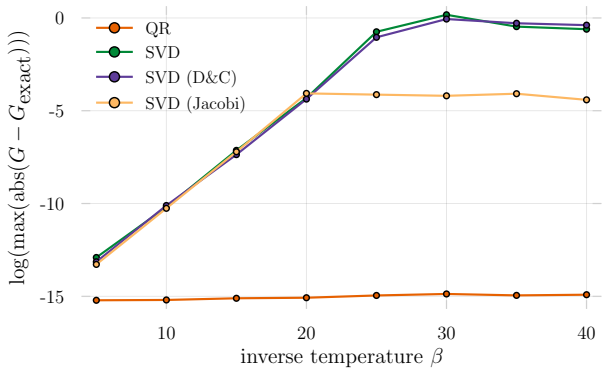

FIG. 4. **Accuracy of the Green's function** obtained from stabilized computations using the listed matrix decompositions and the inversion scheme `inv_one_plus`, Eq. (12).

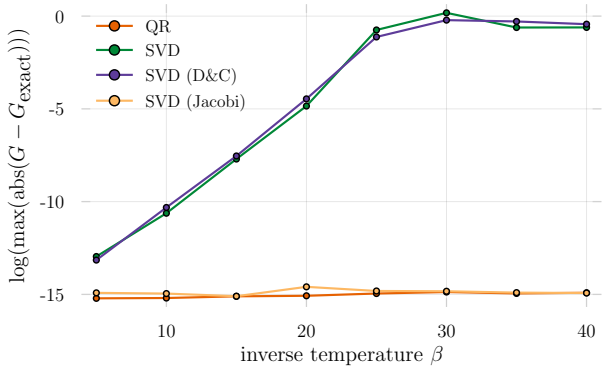

FIG. 5. **Accuracy of the Green's function** obtained from stabilized computations using the listed matrix decompositions and the careful inversion scheme `inv_one_plus_loh`, Eq. 13.

At high temperatures, 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 reproduces G_{exact} reliably. It has the highest accuracy by a large margin, followed by the Jacobi SVD as the best of the SVD methods, which all 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 improve the accuracy of the Jacobi SVD dramatically while the deviations seen for the other two SVD based schemes are still of order unity.

In Figs. 6, 7 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`.

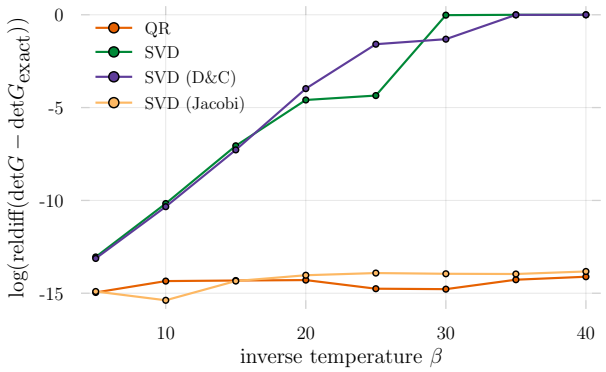

FIG. 6. **Accuracy of the determinant** of the equal-time Green's function obtained from stabilized computations using the listed matrix decompositions and the inversion scheme `inv_one_plus`, Eq. 12.

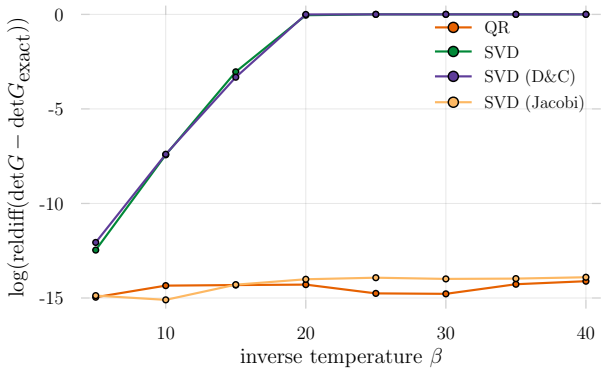

FIG. 7. **Accuracy of the determinant** of the equal-time Green's function obtained from stabilized computations using the listed matrix decompositions and `inv_one_plus_loh`, Eq. 13.

These findings suggest that only the QR decomposition, independent of the inversion procedure, or the Jacobi SVD in combination with the more careful inversion scheme `inv_one_plus_loh` is suited for computing both the equal time Green's function and it's determinant reliably.

B. Efficiency

IV. TIME-DISPLACED GREEN'S FUNCTION

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

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

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[5,6]

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

where T represents time ordering.

More explicitly this reads

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

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

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

Similar to Sec. II, 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` in `StableDQMC.jl`),

$$\begin{aligned} G(\tau_1, \tau_2) &= [U_L D_L X_L + U_R D_R X_R]^{-1} \\ &= [U_L \underbrace{(D_L X_L X_R^{-1} + U_L^\dagger U_R D_R)}_{udx} X_R]^{-1} \\ &= [(U_L u) d^{-1} (x X_R)]^{-1} \qquad (19) \\ &= U_r D_r X_r, \qquad (20) \end{aligned}$$

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` in `StableDQMC.jl`),[7]

$$\begin{aligned} G(\tau_1, \tau_2) &= [U_L D_L X_L + U_R D_R X_R]^{-1} \\ &= [U_L D_{Lm} D_{Lp} X_L + U_R D_{Rm} D_{Rp} X_R]^{-1} \\ &= \left[U_L D_{Lp} \underbrace{\left(\frac{D_{Lm}}{D_{Rp}} X_L X_R^{-1} + U_L^\dagger U_R \frac{D_{Rm}}{D_{Lp}} \right)}_{udx} X_R D_{Rp} \right]^{-1} \\ &= X_R^{-1} \underbrace{\frac{1}{D_{Rp}} [udx]^{-1} \frac{1}{D_{Lp}}}_{udx} U_L^\dagger \qquad (21) \\ &= U_r D_r X_r, \qquad (22) \end{aligned}$$

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

A. Accuracy

OPT: Mention Hirsch method[13]

[1] R Blankenbecler, D J Scalapino, and R L Sugar, "Monte Carlo calculations of coupled boson-fermion systems. I," Physical Review D **24**, 2278–2286 (1981).

[2] J. E. Hirsch, "Discrete Hubbard-Stratonovich transformation for fermion lattice models," Physical Review B **28**, 4059–4061 (1983).

[3] David Goldberg, "What every computer scientist should know about floating-point arithmetic," ACM Comput. Surv. **23**, 5–48 (1991).

[4] Note that we intentionally do not discuss the option to calculate B^M as UD^MX since typically the system will be interacting and the B matrices in the product chain will differ.

[5] Raimundo R dos Santos, "Introduction to quantum Monte Carlo simulations for fermionic systems," Brazilian Journal of Physics **33**, 36–54 (2003).

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

[7] 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**, 1319–1327 (2005).

[8] 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.

[9] 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).

[10] 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`.

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

[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. 7. In fact, in an optimal implementation the computation of the acceptance rate is $O(1)$ rather than $O(N^3)$.

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

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