The *ALF* (*A*lgorithms for *L*attice *F*ermions) project release 1.2

Documentation for the auxiliary-field quantum Monte Carlo code.

Martin Bercx, Florian Goth, Johannes S. Hofmann, Fakher F. Assaad

February 20, 2020

Contents

1 Introduction

1.1 Motivation

The auxiliary-field quantum Monte Carlo (QMC) approach is the algorithm of choice to simulate thermodynamic properties of a variety of correlated electron systems in the solid state and beyond [1, 2, 3, 4, 5, 6]. Apart from the physics of the canonical Hubbard model [7, 8], the topics one can investigate in detail include correlation effects in the bulk and on surfaces of topological insulators [9, 10], quantum phase transitions between Dirac fermions and insulators [11, 12, 13, 14, 15], deconfined quantum critical points [16, 17], topologically ordered phases [17], heavy fermion systems [18, 19], nematic [20] and magnetic [**?**] quantum phase transitions in metals, antiferromagnetism in metals [21], superconductivity in spin-orbit split bands [22], $SU(N)$ symmetric models [23, 24], long-ranged Coulomb interactions in graphene systems [25, 26], cold atomic gases [27], low energy nuclear physics [28], entanglement entropies and spectra [29, 30, 31, 32], etc. This ever-growing list of topics is based on algorithmic progress and on recent symmetry-related insights [33, 34, 35, 36] that lead to formulations free of the negative sign problem for a number of model systems with very rich phase diagrams.

Auxiliary-field methods can be formulated in very different ways. The fields define the configuration space $\mathcal{C}$. They can stem from the Hubbard-Stratonovich (HS) [37] transformation required to decouple the many-body interacting term into a sum of non-interacting problems, or they can correspond to bosonic modes with predefined dynamics such as phonons or gauge fields. In all cases, the result is that the grand-canonical partition function takes the form,

$$Z = \mathrm{Tr}\left(e^{-\beta\hat{\mathcal{H}}}\right) = \sum_{\mathcal{C}} e^{-S(\mathcal{C})}, \tag{1}$$

where β corresponds to the inverse temperature and S is the action of non-interacting fermions subject to a space-time fluctuating auxiliary field. The high-dimensional integration over the fields is carried out stochastically. In this formulation of many body quantum systems, there is no reason for the action to be a real number. Thereby $e^{-S(\mathcal{C})}$ cannot be interpreted as a weight. To circumvent this problem one can adopt re-weighting schemes and sample $|e^{-S(\mathcal{C})}|$. This invariably leads to the so-called *negative sign problem*, with the associated exponential computational scaling in system size and inverse temperature [38]. The sign problem is formulation dependent and, as mentioned above, there has been tremendous progress at identifying an increasing number of models not affected by the negative sign problem which cover a rich domain of collective emergent phenomena. For continuous fields, the stochastic integrations can be carried out with Langevin dynamics or hybrid methods [39]. However, for many problems one can get away with discrete fields [40]. In this case, Monte Carlo importance sampling will often be put to use [41]. We note that due to the non-locality of the fermion determinant, see below, cluster updates, such as in the loop or stochastic series expansion algorithms for quantum spin systems [42, 43, 44], are hard to formulate for this class of problems. The search for efficient updating schemes that enable to move quickly within the configuration space, defines ongoing challenges.

Formulations do not differ only by the choice of the fields, continuous or discrete, and the sampling strategy, but also by the formulation of the action itself. For a given field configuration, integrating out fermionic degrees of freedom generically leads to a fermionic determinant of dimension βN where N is the volume of the system. Working with this determinant leads to the Hirsch-Fye approach [45] and the computational effort scales[1] as $\mathcal{O}\left(\beta N\right)^3$. The Hirsch-Fye algorithm is the method of choice for impurity problems, but has generically been outperformed by a class of so-called continuous-time quantum Monte Carlo approaches [46, 47, 48]. One key point of continuous-time methods is that they are action based and thereby allow one to handle the retarded interactions obtained when integrating out fermion or boson baths. In high dimensions and/or at low temperatures, the cubic scaling originating from the fermionic determinant is expensive. To circumvent this, the hybrid Monte-Carlo approach [49, 5] expresses the fermionic determinant in terms of a Gaussian integral thereby introducing a new variable in the Monte Carlo integration. The resulting algorithm is the method of choice for lattice gauge theories in 3+1 dimensions and has been used to provide *ab initio* estimates of light hadron masses starting from quantum chromodynamics [50]. The approach we consider here lies between the above two *extremes*. We keep the fermionic determinant, but formulate the problem so as to work only with $N \times N$ matrices. This Blankenbecler, Scalapino, Sugar (BSS) algorithm scales linearly in imaginary time β, but remains cubic in the volume N. Furthermore, the algorithm can be formulated either in a projective manner [3, 4],

[1] Here we implicitly assume the absence of negative sign problem

adequate to obtain zero temperature properties in the canonical ensemble, or at finite temperatures in the grand-canonical ensemble [2].

The aim of the ALF project is to introduce a general formulation of the finite-temperature auxiliary-field QMC method with discrete fields so as enable one to promptly play with different model Hamiltonians at minimal programming cost. In this documentation, we summarize the essential aspects of the auxiliary-field QMC approach, and refer the reader to Refs. [51, 6] for complete reviews. In the following we show in all details how to implement a variety of models, run the code, and produce results for equal-time and time-displaced correlation functions. The program code is written in Fortran according to the 2003 standard and is able to natively utilize MPI for massively parallel runs on today's supercomputing systems.

1.2 Definition of the Hamiltonian

The first and most fundamental part of the project is to define a general Hamiltonian which can accommodate a large class of models. Our approach is to express the model as a sum of one-body terms, a sum of two-body terms each written as a perfect square of a one body term, as well as a one-body term coupled to an Ising field with dynamics to be specified by the user. Writing the interaction in terms of sums of perfect squares allows us to use generic forms of discrete approximations to the HS transformation [52, 53]. Symmetry considerations are imperative to increase the speed of the code. We therefore include a *color* index reflecting an underlying $SU(N)$ color symmetry as well as a *flavor* index reflecting the fact that after the HS transformation, the fermionic determinant is block diagonal in this index.

The class of solvable models includes Hamiltonians $\hat{\mathcal{H}}$ that have the following general form:

$$\hat{\mathcal{H}} = \hat{\mathcal{H}}_T + \hat{\mathcal{H}}_V + \hat{\mathcal{H}}_I + \hat{\mathcal{H}}_{0,I}\,, \text{ where} \tag{2}$$

$$\hat{\mathcal{H}}_T = \sum_{k=1}^{M_T} \sum_{\sigma=1}^{N_{\mathrm{col}}} \sum_{s=1}^{N_{\mathrm{fl}}} \sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}^{\dagger}_{x\sigma s} T^{(ks)}_{xy} \hat{c}_{y\sigma s} \equiv \sum_{k=1}^{M_T} \hat{T}^{(k)}\,, \tag{3}$$

$$\hat{\mathcal{H}}_V = \sum_{k=1}^{M_V} U_k \left\{ \sum_{\sigma=1}^{N_{\mathrm{col}}} \sum_{s=1}^{N_{\mathrm{fl}}} \left[\left(\sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}^{\dagger}_{x\sigma s} V^{(ks)}_{xy} \hat{c}_{y\sigma s} \right) + \alpha_{ks} \right] \right\}^2 \equiv \sum_{k=1}^{M_V} U_k \left(\hat{V}^{(k)} \right)^2\,, \tag{4}$$

$$\hat{\mathcal{H}}_I = \sum_{k=1}^{M_I} \hat{Z}_k \left(\sum_{\sigma=1}^{N_{\mathrm{col}}} \sum_{s=1}^{N_{\mathrm{fl}}} \sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}^{\dagger}_{x\sigma s} I^{(ks)}_{xy} \hat{c}_{y\sigma s} \right) \equiv \sum_{k=1}^{M_I} \hat{Z}_k \hat{I}^{(k)}\,. \tag{5}$$

The indices and symbols used above have the following meaning:

- The number of fermion *flavors* is set by N_{fl}. After the HS transformation, the action will be block diagonal in the flavor index.
- The number of fermion *colors* is set[2] by N_{col}. The Hamiltonian is invariant under $SU(N_{\mathrm{col}})$ rotations.
- N_{dim} is the total number of spacial vertices: $N_{\mathrm{dim}} = N_{\text{unit cell}} N_{\mathrm{orbital}}$, where $N_{\text{unit cell}}$ is the number of unit cells of the underlying Bravais lattice and N_{orbital} is the number of (spacial) orbitals per unit cell.
- The indices x and y label lattice sites where $x, y = 1, \cdots, N_{\mathrm{dim}}$.
- Therefore, the matrices $\boldsymbol{T}^{(ks)}$, $\boldsymbol{V}^{(ks)}$ and $\boldsymbol{I}^{(ks)}$ are of dimension $N_{\mathrm{dim}} \times N_{\mathrm{dim}}$.
- The number of interaction terms is labeled by M_V and M_I. $M_T > 1$ would allow for a checkerboard decomposition.
- $\hat{c}^{\dagger}_{y\sigma s}$ is a second-quantized operator that creates an electron in a Wannier state centered around lattice site y, with color σ, and flavor index s. The operators satisfy the anti-commutation relations:

$$\left\{ \hat{c}^{\dagger}_{y\sigma s}, \hat{c}_{y'\sigma' s'} \right\} = \delta_{x,x'} \delta_{s,s'} \delta_{\sigma,\sigma'}, \quad \text{and} \quad \left\{ \hat{c}_{y\sigma s}, \hat{c}_{y'\sigma' s'} \right\} = 0. \tag{6}$$

The Ising part of the general Hamiltonian (2) is $\hat{\mathcal{H}}_{0,I} + \hat{\mathcal{H}}_I$ and has the following properties:

[2] Note that in the code $N_{\mathrm{col}} \equiv$ `N_SUN`.

- $\hat{Z}_k$ couples to a general one-body term. It is a general operator, such as the Ising spin operator corresponding to the Pauli matrix $\hat{\sigma}_z$ or the position operator.
- The dynamics of the Ising spins is given by $\hat{\mathcal{H}}_{0,I}$. This term is not specified here; it has to be specified by the user and becomes relevant when the Monte Carlo update probability is computed in the code (see Sec. 5.4 for an example).

Note that the matrices $\boldsymbol{T}^{(ks)}$, $\boldsymbol{V}^{(ks)}$ and $\boldsymbol{I}^{(ks)}$ explicitly depend on the flavor index s but not on the color index σ. The color index σ only appears in the second quantized operators such that the Hamiltonian is manifestly $SU(N_{\mathrm{col}})$ symmetric. We also require the matrices $\boldsymbol{T}^{(ks)}$, $\boldsymbol{V}^{(ks)}$ and $\boldsymbol{I}^{(ks)}$ to be Hermitian.

1.3 Outline

In order to use the code, a minimal understanding of the algorithm is necessary. In Sec. 2, we go very briefly through the steps required to formulate the many-body, imaginary-time propagation in terms of a sum over HS and Ising fields of one-body, imaginary-time propagators. The user has to provide this one-body, imaginary-time propagator for a given configuration of HS and Ising fields. We also discuss the Monte Carlo updates, the strategies for numerical stabilization of the code, as well as the Monte Carlo sampling.

Section 3 is devoted to the data structures that are needed to implement the model, as well as to the input and output file structure. The data structure includes an `Operator` type to optimally work with sparse Hermitian matrices, a `Lattice` type to define one- and two-dimensional Bravais lattices, and two `Observable` types to handle scalar observables (e.g., total energy) and equal-time or time-displaced two-point correlation functions (e.g., spin-spin correlations).

The Monte Carlo simulation and the associated data analysis are separated processes: the QMC run dumps the results of *bins* sequentially into files which are then analyzed by analysis programs. In Sec. 3.5, we provide a brief description of the analysis programs for our observable types. The analysis programs allow for omitting a given number of initial bins in order to account for warm-up times. Also, a rebinning analysis is included to account for, *a posteriori*, long autocorrelation times. Finally, Sec. 3.6 provides all the necessary details to compile and run the code.

In Sec. 5, we give explicit examples on how to use the code for the Hubbard model on square and honeycomb lattices, for different choices of the Hubbard-Stratonovich transformation (see Secs. 5.1, 5.2 and 5.3), as well as for the Hubbard model on a square lattice coupled to a transverse Ising field (see Sec. 5.4). Our implementation is rather general, allowing for a variety of other models to be simulated and, in Sec. 6, further examples are given, such as the Kondo lattice and the $SU(N)$ symmetric Hubbard-Heisenberg models.

Finally, in Sec. 7 we list a number of features being considered for future releases of the ALF program package.

2 Auxiliary Field Quantum Monte Carlo

2.1 Formulation of the method

Our aim is to compute observables for the general Hamiltonian (2) in thermodynamic equilibrium as described by the grand-canonical ensemble. We show below how the grand-canonical partition function can be rewritten as

$$Z = \mathrm{Tr}\left(e^{-\beta\hat{\mathcal{H}}}\right) = \sum_C e^{-S(C)} + \mathcal{O}(\Delta\tau^2) \tag{7}$$

and define the space of configurations C. Note that the chemical potential term is already included in the definition of the one-body term $\hat{\mathcal{H}}_T$, see Eq. (3), of the general Hamiltonian.

The outline of this section is as follows. First, we derive the detailed form of the partition function and outline the computation of observables (Sec. 2.1.1 - 2.1.3). Next, we present a number of update strategies, namely local updates, global updates, and parallel tempering (Sec. 2.2). We also describe the measures we have implemented to make the code numerically stable (Sec. 2.3). Finally, we discuss the autocorrelations and associated time scales during the Monte Carlo sampling process (Sec. 2.4).

The essential ingredients of the auxiliary-field quantum Monte Carlo implementation in the ALF package are the following:

- We discretize the imaginary time propagation: $\beta = \Delta\tau L_{\text{Trotter}}$. Generically this introduces a systematic Trotter error of $\mathcal{O}(\Delta\tau)^2$ [54]. We note that there has been considerable effort at getting rid of the Trotter systematic error and to formulate a genuine continuous-time BSS algorithm [55]. To date, efforts in this direction are based on a CT-AUX type formulation [56, 57] and face two issues. The first issue is that they are restricted to a class of models with Hubbard-type interactions

$$(\hat{n}_i - 1)^2 = (\hat{n}_i - 1)^4 \,, \tag{8}$$

such that the basic CT-AUX equation [58]

$$1 + \frac{U}{K}(\hat{n}_i - 1)^2 = \frac{1}{2}\sum_{s=\pm 1} e^{\alpha s(\hat{n}_i - 1)} \quad \text{with} \quad \frac{U}{K} = \cosh(\alpha) - 1 \quad \text{and} \quad K \in \mathbb{R} \tag{9}$$

holds. The second issue is that in the continuous-time approach it is hard to formulate a computationally efficient algorithm. Given this situation it turns out that the multi-grid method [59, 60, 61] is an efficient scheme to extrapolate to small imaginary-time steps so as to eliminate the Trotter systematic error if required.

- Having isolated the two-body term, we use the discrete HS transformation [52, 53]:

$$e^{\Delta\tau\lambda\hat{A}^2} = \sum_{l=\pm 1,\pm 2} \gamma(l) e^{\sqrt{\Delta\tau\lambda}\eta(l)\hat{A}} + \mathcal{O}(\Delta\tau^4) \,, \tag{10}$$

where the fields η and γ take the values:

$$\begin{aligned} \gamma(\pm 1) &= 1 + \sqrt{6}/3, \quad \eta(\pm 1) = \pm\sqrt{2\left(3 - \sqrt{6}\right)} \,, \\ \gamma(\pm 2) &= 1 - \sqrt{6}/3, \quad \eta(\pm 2) = \pm\sqrt{2\left(3 + \sqrt{6}\right)} \,. \end{aligned} \tag{11}$$

Since the Trotter error is already of order $(\Delta\tau^2)$ per time slice, this transformation is next to exact.

- $\hat{Z}_k$ can stand for a variety of operators, such as the Pauli matrix $\hat{\sigma}_z$, in which case the Ising spins take the values $s_k = \pm 1$, or the position operator – such that $\hat{Z}_k|\phi\rangle = \phi|\phi\rangle$, with ϕ a real number.

- From the above it follows that the Monte Carlo configuration space C is given by the combined spaces of Ising spin configurations and of HS discrete field configurations:

$$C = \{s_{i,\tau}, l_{j,\tau} \text{ with } i = 1 \cdots M_I,\ j = 1 \cdots M_V,\ \tau = 1 \cdots L_{\text{Trotter}}\} \,. \tag{12}$$

Here, the HS fields take the values $l_{j,\tau} = \pm 2, \pm 1$ and the Ising spins $s_{i,\tau}$ may, for instance, be a continuous real field or, if $\hat{Z}_k = \hat{\sigma}_z$, be restricted to ± 1.

2.1.1 The partition function

With the above, the partition function of the model (2) can be written as follows.

$$\begin{aligned} Z &= \text{Tr}\left(e^{-\beta\hat{\mathcal{H}}}\right) \\ &= \text{Tr}\left[e^{-\Delta\tau\hat{\mathcal{H}}_{0,I}} \prod_{k=1}^{M_V} e^{-\Delta\tau U_k\left(\hat{V}^{(k)}\right)^2} \prod_{k=1}^{M_I} e^{-\Delta\tau\hat{\sigma}_k\hat{I}^{(k)}} \prod_{k=1}^{M_T} e^{-\Delta\tau\hat{T}^{(k)}}\right]^{L_{\text{Trotter}}} + \mathcal{O}(\Delta\tau^2) \\ &= \sum_C \left(\prod_{k=1}^{M_V}\prod_{\tau=1}^{L_{\text{Trotter}}} \gamma_{k,\tau}\right) e^{-S_{0,I}(\{s_{i,\tau}\})} \times \\ &\quad \text{Tr}_{\text{F}}\left\{\prod_{\tau=1}^{L_{\text{Trotter}}}\left[\prod_{k=1}^{M_V} e^{\sqrt{-\Delta\tau U_k}\eta_{k,\tau}\hat{V}^{(k)}} \prod_{k=1}^{M_I} e^{-\Delta\tau s_{k,\tau}\hat{I}^{(k)}} \prod_{k=1}^{M_T} e^{-\Delta\tau\hat{T}^{(k)}}\right]\right\} + \mathcal{O}(\Delta\tau^2) \,. \end{aligned} \tag{13}$$

In the above, the trace Tr runs over the Ising spins as well as over the fermionic degrees of freedom, and Tr_{F} only over the fermionic Fock space. $S_{0,I}(\{s_{i,\tau}\})$ is the action corresponding to the Ising Hamiltonian,

and is only dependent on the Ising spins so that it can be pulled out of the fermionic trace. We have adopted the short hand notation $\eta_{k,\tau} = \eta(l_{k,\tau})$ and $\gamma_{k,\tau} = \gamma(l_{k,\tau})$. At this point, and since for a given configuration C we are dealing with a free propagation, we can integrate out the fermions to obtain a determinant:

$$\begin{aligned}
&\mathrm{Tr}_{\mathrm{F}}\left\{\prod_{\tau=1}^{L_{\mathrm{Trotter}}}\left[\prod_{k=1}^{M_V} e^{\sqrt{-\Delta\tau U_k}\eta_{k,\tau}\hat{V}^{(k)}}\prod_{k=1}^{M_I} e^{-\Delta\tau s_{k,\tau}\hat{I}^{(k)}}\prod_{k=1}^{M_T} e^{-\Delta\tau\hat{T}^{(k)}}\right]\right\} = \\
&\prod_{s=1}^{N_{\mathrm{fl}}}\left[e^{\sum_{k=1}^{M_V}\sum_{\tau=1}^{L_{\mathrm{Trotter}}}\sqrt{-\Delta\tau U_k}\alpha_{k,s}\eta_{k,\tau}}\right]^{N_{\mathrm{col}}} \times \\
&\prod_{s=1}^{N_{\mathrm{fl}}}\left[\det\left(\mathbb{1} + \prod_{\tau=1}^{L_{\mathrm{Trotter}}}\prod_{k=1}^{M_V} e^{\sqrt{-\Delta\tau U_k}\eta_{k,\tau}\boldsymbol{V}^{(ks)}}\prod_{k=1}^{M_I} e^{-\Delta\tau s_{k,\tau}\boldsymbol{I}^{(ks)}}\prod_{k=1}^{M_T} e^{-\Delta\tau\mathbf{T}^{(ks)}}\right)\right]^{N_{\mathrm{col}}},
\end{aligned} \tag{14}$$

where the matrices $\mathbf{T}^{(ks)}$, $\mathbf{V}^{(ks)}$, and $\mathbf{I}^{(ks)}$ define the Hamiltonian [Eq. (2) - (5)]. All in all, the partition function is given by:

$$\begin{aligned}
Z = &\sum_C e^{-S_{0,I}(\{s_{i,\tau}\})}\left(\prod_{k=1}^{M_V}\prod_{\tau=1}^{L_{\mathrm{Trotter}}}\gamma_{k,\tau}\right) e^{N_{\mathrm{col}}\sum_{s=1}^{N_{\mathrm{fl}}}\sum_{k=1}^{M_V}\sum_{\tau=1}^{L_{\mathrm{Trotter}}}\sqrt{-\Delta\tau U_k}\alpha_{k,s}\eta_{k,\tau}} \times \\
&\prod_{s=1}^{N_{\mathrm{fl}}}\left[\det\left(\mathbb{1} + \prod_{\tau=1}^{L_{\mathrm{Trotter}}}\prod_{k=1}^{M_V} e^{\sqrt{-\Delta\tau U_k}\eta_{k,\tau}\boldsymbol{V}^{(ks)}}\prod_{k=1}^{M_I} e^{-\Delta\tau s_{k,\tau}\boldsymbol{I}^{(ks)}}\prod_{k=1}^{M_T} e^{-\Delta\tau\boldsymbol{T}^{(ks)}}\right)\right]^{N_{\mathrm{col}}} + \mathcal{O}(\Delta\tau^2) \\
\equiv &\sum_C e^{-S(C)} + \mathcal{O}(\Delta\tau^2)\,.
\end{aligned} \tag{15}$$

In the above, one notices that the weight factorizes in the flavor index. The color index raises the determinant to the power N_{col}. This corresponds to an explicit $SU(N_{\mathrm{col}})$ symmetry for each configuration. This symmetry is manifest in the fact that the single particle Green functions are color independent, again for each given configuration C.

2.1.2 Observables

In the auxiliary-field QMC approach, the single-particle Green function plays a crucial role. It determines the Monte Carlo dynamics and is used to compute observables:

$$\langle\hat{O}\rangle = \frac{\mathrm{Tr}\left[e^{-\beta\hat{H}}\hat{O}\right]}{\mathrm{Tr}\left[e^{-\beta\hat{H}}\right]} = \sum_C P(C)\langle\langle\hat{O}\rangle\rangle_{(C)}, \text{ with } P(C) = \frac{e^{-S(C)}}{\sum_C e^{-S(C)}}\,, \tag{16}$$

and $\langle\langle\hat{O}\rangle\rangle_{(C)}$ denotes the observed value of $\hat{O}$ for a given configuration C. For a given configuration C one can use Wick's theorem to compute $O(C)$ from the knowledge of the single-particle Green function:

$$G(x,\sigma,s,\tau|x',\sigma',s',\tau') = \langle\langle\mathcal{T}\hat{c}_{x\sigma s}(\tau)\hat{c}^{\dagger}_{x'\sigma's'}(\tau')\rangle\rangle_C \tag{17}$$

where $\mathcal{T}$ corresponds to the imaginary-time ordering operator. The corresponding equal time quantity reads,

$$G(x,\sigma,s,\tau|x',\sigma',s',\tau) = \langle\langle\hat{c}_{x\sigma s}(\tau)\hat{c}^{\dagger}_{x'\sigma's'}(\tau)\rangle\rangle_C. \tag{18}$$

Since, for a given HS field, translation invariance in imaginary-time is broken, the Green function has an explicit τ and τ' dependence. On the other hand it is diagonal in the flavor index, and independent of the color index. The latter reflects the explicit $SU(N)$ color symmetry present at the level of individual HS configurations. As an example, one can show that the equal-time Green function at $\tau = 0$ reads [6]:

$$G(x,\sigma,s,0|x',\sigma,s,0) = \left(\mathbb{1} + \prod_{\tau=1}^{L_{\mathrm{Trotter}}}\boldsymbol{B}_{\tau}^{(s)}\right)^{-1}_{x,x'} \tag{19}$$

with

$$\boldsymbol{B}_{\tau}^{(s)} = \prod_{k=1}^{M_V} e^{\sqrt{-\Delta\tau U_k}\eta_{k,\tau}\boldsymbol{V}^{(ks)}} \prod_{k=1}^{M_I} e^{-\Delta\tau s_{k,\tau}\boldsymbol{I}^{(ks)}} \prod_{k=1}^{M_T} e^{-\Delta\tau \boldsymbol{T}^{(ks)}}. \tag{20}$$

To compute equal-time, as well as time-displaced observables, one can make use of Wick's theorem. A convenient formulation of this theorem for QMC simulations reads:

$$\langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_1}(\tau_1) c_{\underline{x}'_1}(\tau'_1) \cdots c^{\dagger}_{\underline{x}_n}(\tau_n) c_{\underline{x}'_n}(\tau'_n)\rangle\rangle_C =$$
$$\det \begin{bmatrix} \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_1}(\tau_1) c_{\underline{x}'_1}(\tau'_1)\rangle\rangle_C & \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_1}(\tau_1) c_{\underline{x}'_2}(\tau'_2)\rangle\rangle_C & \cdots & \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_1}(\tau_1) c_{\underline{x}'_n}(\tau'_n)\rangle\rangle_C \\ \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_2}(\tau_2) c_{\underline{x}'_1}(\tau'_1)\rangle\rangle_C & \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_2}(\tau_2) c_{\underline{x}'_2}(\tau'_2)\rangle\rangle_C & \cdots & \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_2}(\tau_2) c_{\underline{x}'_n}(\tau'_n)\rangle\rangle_C \\ \vdots & \vdots & \ddots & \vdots \\ \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_n}(\tau_n) c_{\underline{x}'_1}(\tau'_1)\rangle\rangle_C & \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_n}(\tau_n) c_{\underline{x}'_2}(\tau'_2)\rangle\rangle_C & \cdots & \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_n}(\tau_n) c_{\underline{x}'_n}(\tau'_n)\rangle\rangle_C \end{bmatrix}. \tag{21}$$

Here, we have defined the super-index $\underline{x} = \{x, \sigma, s\}$. In the subroutines `Obser` and `ObserT` of the module `Hamiltonian_Examples.F90` (see Sec. 3.2) the user is provided with the equal-time and time-displaced correlation functions. Using the above formulation of Wick's theorem, arbitrary correlation functions can be computed. We note, however, that the program is limited to the calculation of observables that contain only two different imaginary times.

2.1.3 Reweighting and the sign problem

In general, the action $S(C)$ will be complex, thereby inhibiting a direct Monte Carlo sampling of $P(C)$. This leads to the infamous sign problem. The sign problem is formulation dependent and as noted above, much progress has been made at understanding the class of models that can be formulated without encountering this problem [33, 34, 35, 36]. When the average sign is not too small, we can nevertheless compute observables within a reweighting scheme. Here we adopt the following scheme. First note that the partition function is real such that:

$$Z = \sum_C e^{-S(C)} = \sum_C \overline{e^{-S(C)}} = \sum_C \mathrm{Re}\left[e^{-S(C)}\right]. \tag{22}$$

Thereby[3] and with the definition

$$\mathrm{sign}\ (C) = \frac{\mathrm{Re}\left[e^{-S(C)}\right]}{\left|\mathrm{Re}\left[e^{-S(C)}\right]\right|}, \tag{23}$$

the computation of the observable [Eq. (16)] is re-expressed as follows:

$$\begin{aligned} \langle \hat{O} \rangle &= \frac{\sum_C e^{-S(C)} \langle\langle \hat{O} \rangle\rangle_{(C)}}{\sum_C e^{-S(C)}} \\ &= \frac{\sum_C \mathrm{Re}\left[e^{-S(C)}\right] \frac{e^{-S(C)}}{\mathrm{Re}\left[e^{-S(C)}\right]} \langle\langle \hat{O} \rangle\rangle_{(C)}}{\sum_C \mathrm{Re}\left[e^{-S(C)}\right]} \\ &= \frac{\left\{\sum_C \left|\mathrm{Re}\left[e^{-S(C)}\right]\right| \ \mathrm{sign}\ (C) \frac{e^{-S(C)}}{\mathrm{Re}\left[e^{-S(C)}\right]} \langle\langle \hat{O} \rangle\rangle_{(C)}\right\} / \sum_C \left|\mathrm{Re}\left[e^{-S(C)}\right]\right|}{\left\{\sum_C \left|\mathrm{Re}\left[e^{-S(C)}\right]\right| \ \mathrm{sign}\ (C)\right\} / \sum_C \left|\mathrm{Re}\left[e^{-S(C)}\right]\right|} \\ &= \frac{\left\langle \mathrm{sign}\ \frac{e^{-S}}{\mathrm{Re}[e^{-S}]} \langle\langle \hat{O} \rangle\rangle \right\rangle_{\overline{P}}}{\langle \mathrm{sign} \rangle_{\overline{P}}}. \end{aligned} \tag{24}$$

The average sign is

$$\langle \mathrm{sign} \rangle_{\overline{P}} = \frac{\sum_C \left|\mathrm{Re}\left[e^{-S(C)}\right]\right| \ \mathrm{sign}\ (C)}{\sum_C \left|\mathrm{Re}\left[e^{-S(C)}\right]\right|}, \tag{25}$$

[3] The attentive reader will have noticed that for arbitrary Trotter decompositions, the imaginary time propagator is not necessarily Hermitian. Thereby, the above equation is correct only up to corrections stemming from the controlled Trotter systematic error.

and we have $\langle \text{sign} \rangle_{\overline{P}} \in \mathbb{R}$ per definition. The Monte Carlo simulation samples the probability distribution

$$\overline{P}(C) = \frac{\left| \text{Re}\left[e^{-S(C)} \right] \right|}{\sum_C \left| \text{Re}\left[e^{-S(C)} \right] \right|} \,. \tag{26}$$

such that the nominator and denominator of Eq. (24) can be computed.

The negative sign problem is still an issue because the average sign is a ratio of two partition functions and one can argue that

$$\langle \text{sign} \rangle_{\overline{P}} \propto e^{-\Delta N \beta}, \tag{27}$$

where Δ is intensive positive quantity and $N\beta$ denotes the Euclidean volume. In a Monte Carlo simulation the error scales as $1/\sqrt{T_{CPU}}$ where T_{CPU} corresponds to the computational time. Since the error on the average sign has to be much smaller than the average sign itself, one sees that:

$$T_{CPU} \gg e^{2\Delta N \beta}. \tag{28}$$

Two comments are in order. First, the presence of a sign problem invariably leads to an exponential increase of CPU time as a function of the Euclidean volume. And second, Δ is formulation dependent. For instance, at finite doping, the SU(2) invariant formulation of the Hubbard model presented in Sec. 5.1 has a much more severe sign problem than the formulation presented in Sec. 5.2 where the HS field couples to the z-component of the magnetization.

2.2 Updating schemes

The program allows for different types of updating schemes, which are described below and summarized in Tab. 1. In all of them, given a configuration C, we propose a new one, C', with probability $T_0(C \to C')$ and accept it according to the Metropolis-Hastings acceptance-rejection probability,

$$P(C \to C') = \min\left(1, \frac{T_0(C' \to C) W(C')}{T_0(C \to C') W(C)}\right), \tag{29}$$

so as to guarantee the stationarity condition. Here, $W(C) = \left| \text{Re}\left[e^{-S(C)} \right] \right|$.

Updating schemes	Type	Description
`Propose_S0`	Logical	If true, proposes sequential local moves according to the probability e^{-S_0}.
`Global_tau_moves`	Logical	This option allows to carry out global moves on a single time slice. For a given time slice the user can define which part of the operator string is to be computed sequentially. This is specified by the variable `N_sequential_start` and `Nt_sequential_end`. A number of `N_tau_Global` user-defined global moves on the given time slice will then be carried out.
`Global_moves`	Logical	If true, allows for global moves in space and time. A user-defined number `N_Global` of global moves in space and time will be carried out at the end of each sweep.
`TEMPERING`	Compiling option	Requires MPI and runs the code in a parallel tempering mode, also see Sec. 2.2.5.

Table 1: Variables required to control the updating scheme. Per default the program carries out sequential, single spin-flip sweeps, and logical variables are set to `false`.

2.2.1 Sampling of e^{-S_0}

This section refers to single spin-flip updates in the presence of a non-vanishing Ising action $S_0(C)$. This sampling scheme is used if the logical variable `Propose_S0` is set to `true`.

Consider an Ising spin at space-time i, τ in the configuration C. Flipping this spin will generate the configuration C' and we will propose the move according to

$$T_0(C \to C') = \frac{e^{-S_0(C')}}{e^{-S_0(C')} + e^{-S_0(C)}} = 1 - \frac{1}{1 + e^{-S_0(C')}/e^{-S_0(C)}} \tag{30}$$

Note that the function `S0` in the `Hamitonian_example.F90` module computes precisely the ratio $e^{-S_0(C')}/e^{-S_0(C)}$, therefore $T_0(C \to C')$ is obtained without any additional calculation. The proposed move is accepted with the probability:

$$P(C \to C') = \min\left(1, \frac{e^{-S_0(C)} W(C')}{e^{-S_0(C')} W(C)}\right). \tag{31}$$

Note that, as can be seen from Eq. (15), the bare action $S_0(C)$ determining the dynamics of the Ising spin in the absence of coupling to the fermions does not enter the Metropolis acceptance-rejection step.

2.2.2 Sequential single spin flips

The program adopts per default a sequential, single spin-flip strategy. It will visit sequentially each Hubbard-Stratonovich field in the space-time operator list and propose a spin flip. Consider the Ising spin $s_{i,\tau}$. We will flip it with probability 1, such that for this local move the proposal matrix is symmetric. If we are considering the Hubbard-Stratonovich field $l_{i,\tau}$ we will propose with probability 1/3 one of the other three possible fields. Again, for this local move, the proposal matrix is symmetric. Hence in both cases we will accept or reject the move according to

$$P(C \to C') = \min\left(1, \frac{W(C')}{W(C)}\right). \tag{32}$$

This default updating scheme can be overruled by, e.g., setting `Global_tau_moves` to `true` and not setting `Nt_sequential_start` and `Nt_sequential_end` (see Sec. 3.3.1). It is also worth noting that this type of sequential spin-flip updating does not satisfy detailed balance, but rather the more fundamental stationarity condition [41].

2.2.3 Global updates in space

This option allows one to carry out user-defined global moves on a single time slice. This option is enabled by setting the logical variable `Global_tau_moves` to `true`. To set the stage we recall that the propagation over a time step $\Delta\tau$ (see Eq. 20) can be written as:

$$e^{-V_{M_I+M_V}(s_{M_I+M_V,\tau})} \cdots e^{-V_1(s_{1,\tau})} \prod_{k=1}^{M_T} e^{-\Delta\tau \boldsymbol{T}^{(k)}} \tag{33}$$

where $e^{-V_n(s_n)}$ denotes one element of the operator list containing the HS fields. One can provide an interval of indices, [`Nt_sequential_start`, `Nt_sequential_end`], in which the operators will be updated sequentially. Setting `Nt_sequential_start` $= 1$ and `Nt_sequential_end` $= M_I + M_V$ reproduces the sequential single spin flip strategy of the above section.

The variable `N_tau_Global` sets the number of global moves carried out on each time slice `ntau`. Each global move is generated in the routine `Global_move_tau`, which is provided by the user in the Hamiltonian file. In order to define this move, one specifies the following variables:

- `Flip_length`: An integer stipulating the number of spins to be flipped.
- `Flip_list(1:Flip_length)`: Integer array containing the indices of the operators to be flipped.
- `Flip_value(1:Flip_length)`: `Flip_value(n)` is an integer containing the new value of the HS field for the operator `Flip_list(n)`.
- `T0_Proposal_ratio`: Real number containing the quotient

$$\frac{T_0(C' \to C)}{T_0(C \to C')}, \tag{34}$$

where C' denotes the new configuration obtained by flipping the spins specified in the `Flip_list` array. Since we allow for a stochastic generation of the global move, it may very well be that no change is proposed. In this case, `T0_Proposal_ratio` takes the value 0 upon exit of the routine `Global_move_tau` and no update is carried out.

- `S0_ratio`: Real number containing the ratio $e^{-S_0(C')}/e^{-S_0(C)}$.

2.2.4 Global updates in time and space

The code allows for global updates as well. The user must then provide two additional functions in the module `Hamiltonian_Examples.F90`: `Global_move` and `Delta_S0_global(Nsigma_old)`.

The subroutine `Global_move(T0_Proposal_ratio,nsigma_old,size_clust)` proposes a global move. Its single input is the two-dimensional array `nsigma_old(M_V + M_I, Ltrot)` that contains the full configuration C. On output, the new configuration C', determined by the user, is stored in the two-dimensional array `nsigma`, which is a global variable declared in the module `Hamiltonian`. Like for the global move in space (Sec. 2.2.3), `T0_Proposal_ratio` contains the proposal ratio $T_0(C' \to C)/T_0(C \to C')$. Since we allow for a stochastic generation of the global move, it may very well be that no change is proposed. In this case, `T0_Proposal_ratio` takes the value 0 upon exit, and `nsigma` = `nsigma_old`. The real-valued `size_clust` gives the size of the proposed move (e.g., $\frac{\text{Number of flipped spins}}{\text{Total number of spins}}$). This is used to calculate the average sizes of proposed and accepted moves which will be printed in the `info` file. The variable `size_clust` is not necessary for the simulation, but may help the user to estimate the effectiveness of the global update.

In order to compute the acceptance-rejection ratio, the user must also provide a function `Delta_S0_global(Nsigma_old)` that computes the ratio $e^{-S_0(C')}/e^{-S_0(C)}$. Again, the configuration C' is given by the array `nsigma(M_V + M_I, Ltrot)`, which is a global variable declared in the module `Hamiltonian`.

The variable `N_Global` determines the number of global updates performed per sweep. Note that global updates are expensive, since they require a complete recalculation of the weight.

2.2.5 Parallel tempering

Exchange Monte Carlo [62], or parallel tempering [63], is a possible route to overcome sampling issues in parts of the parameter space. Let h be a parameter which one can vary without altering the configuration space $\{C\}$ and let us assume that for some values of h one encounters sampling problems. For example, in the realm of spin glasses, h could correspond to the inverse temperature. Here at high temperatures the phase space is easily sampled, but at low temperatures simulations get stuck in local minima. For quantum systems, h could trigger a quantum phase transition where sampling issues are encountered, for example, in the ordered phase and not in the disordered one. As its name suggests, parallel tempering carries out in parallel simulations at consecutive values of h: $h_1, h_2, \cdots h_n$, with $h_1 < h_2 < \cdots < h_n$. One will sample the extended ensemble:

$$P([h_1, C_1], [h_2, C_2], \cdots, [h_n, C_n]) = \frac{W(h_1, C_1)W(h_2, C_2)\cdots W(h_n, C_n)}{\sum_{C_1, C_2, \cdots, C_n} W(h_1, C_1)W(h_2, C_2)\cdots W(h_n, C_n)} \tag{35}$$

where $W(h, C)$ corresponds to the weight for for a given value of h and configuration C. Clearly, one can sample $P([h_1, C_1], [h_2, C_2], \cdots, [h_n, C_n])$ by carrying out n independent runs. However, parallel tempering includes the following exchange step:

$$[h_1, C_1], \cdots, [h_i, C_i], [h_{i+1}, C_{i+1}] \cdots, [h_n, C_n] \to [h_1, C_1], \cdots, [h_i, C_{i+1}], [h_{i+1}, C_i] \cdots, [h_n, C_n] \tag{36}$$

which, for a symmetric proposal matrix, will be accepted with probability:

$$\min\left(1, \frac{W(h_i, C_{i+1})W(h_{i+1}, C_i)}{W(h_i, C_i)W(h_{i+1}, C_{i+1})}\right). \tag{37}$$

Thereby, a configuration can meander in parameter space h and explore regions where ergodicity is not an issue. In the context of spin-glasses, a low temperature configuration, stuck in a local minima, can heat up, overcome the potential barrier and then cool down again.

A judicious choice of the values h_i is important to obtain a good acceptance rate for the exchange step. With $W(h, C) = e^{-S(h,C)}$, the distribution of the action S reads:

$$\mathcal{P}(h, S) = \sum_C P(h, C)\delta(S(h, C) - S). \tag{38}$$

A given exchange step can only be accepted if the distributions $\mathcal{P}(h,S)$ and $\mathcal{P}(h+\Delta h,S)$ overlap. For $\langle S\rangle_h < \langle S\rangle_{h+\Delta h}$ one can formulate this requirement as:

$$\langle S\rangle_h + \langle \Delta S\rangle_h \simeq \langle S\rangle_{h+\Delta h} - \langle \Delta S\rangle_{h+\Delta h}, \text{ with } \langle \Delta S\rangle_h = \sqrt{\langle (S-\langle S\rangle_h)^2\rangle_h}. \tag{39}$$

Assuming $\langle \Delta S\rangle_{h+\Delta h} \simeq \langle \Delta S\rangle_h$ and expanding in Δh one obtains:

$$\Delta h \simeq \frac{2\langle \Delta S\rangle_h}{\partial \langle S\rangle_h/\partial h}. \tag{40}$$

The above equation becomes transparent for classical systems with $S(h,C) = hH(C)$. In this case, the above equation reads:

$$\Delta h \simeq 2h\frac{\sqrt{C}}{C+h\langle H\rangle_h}, \text{ with } C = h^2\langle (H-\langle H\rangle_h)^2\rangle_h. \tag{41}$$

Several comments are in order.

i) Let us identify h with the inverse temperature such that C corresponds to the specific heat. This quantity is extensive, as well as the energy, such that $\Delta h \simeq 1/\sqrt{N}$ where N is the system size.

ii) In the proximity of a phase transition the specific heat can diverge, and h must be chosen with particular care.

iii) Since the action is formulation dependent, one expects the acceptance of the exchange move to equally depend upon the formulation.

The quantum Monte Carlo code in the ALF project carries out parallel-tempering runs when compiled with the flag -DTEMPERING, see Sec. 3.6.1.

2.2.6 Langevin dynamics

Neglecting the systematic error origination from the finite imaginary time step, the general from of the action reads:

$$Z = \sum_C e^{-S_{0,I}(\{s_{i,\tau}\})}\left(\prod_{k=1}^{M_V}\prod_{\tau=1}^{L_{\mathrm{Trotter}}}\gamma_{k,\tau}\right) e^{N_{\mathrm{col}}\sum_{s=1}^{N_{\mathrm{fl}}}\sum_{k=1}^{M_V}\sum_{\tau=1}^{L_{\mathrm{Trotter}}}\sqrt{-\Delta\tau U_k}\alpha_{k,s}\eta_{k,\tau}} e^{-S^F(C)} \equiv \sum_C e^{-S(C)} \tag{42}$$

where the fermionic part is given by

$$e^{-S^F(C)} = \prod_{s=1}^{N_{\mathrm{fl}}}\left[\det\left(\mathbb{1} + \prod_{\tau=1}^{L_{\mathrm{Trotter}}}\prod_{k=1}^{M_V} e^{\sqrt{-\Delta\tau U_k}\eta_{k,\tau}\boldsymbol{V}^{(ks)}}\prod_{k=1}^{M_I} e^{-\Delta\tau s_{k,\tau}\boldsymbol{I}^{(ks)}}\prod_{k=1}^{M_T} e^{-\Delta\tau \boldsymbol{T}^{(ks)}}\right)\right]^{N_{\mathrm{col}}}. \tag{43}$$

To formulate both the Langevin dynamics as well as the Hybrid Monte Carlo, we will need to estimate the fermion forces:

$$\frac{\partial S^F(C)}{\partial s_{k,\tau}}. \tag{44}$$

The routine `Langevin_update` in the module `Langevin_update_mod.F90` computes thees forces for a general model and passes them into a user defined routine in the Hamiltonian file `Ham_Langevin_update` where the Langevin update is carried out. Clearly, in this context $s_{k,\tau}$ is a real valued quantity.

Langevin dynamics corresponds to a stochastic differential equation for the fields $\boldsymbol{s} \equiv \{s_{k,\tau}\}$. They acquire a Langevin time t_l and satisfy the stochastic differential equation

$$\boldsymbol{s}(t_l+\delta t_l) = \boldsymbol{s}(t_l) - \frac{\partial S(C)}{\partial \boldsymbol{s}(t_l)}\delta t_l + \sqrt{2\delta t_l}\boldsymbol{\eta}. \tag{45}$$

Here, $\boldsymbol{\eta}$ are independent Gaussian stochastic variables satisfying:

$$\langle \eta_{k,\tau}\rangle_\eta = 0 \text{ and } \langle \eta_{k,\tau}\eta_{k',\tau'}\rangle_\eta = \delta_{k,k'}\delta_{\tau,\tau'} \tag{46}$$

We refer the reader to Ref. [64] for a more in depth introduction to stochastic differential equations. To see that the above indeed produced the desired probability distribution in the long Langevin time limit, we can transform the Langevin equation to the corresponding Fokker-Plank one. Let $P(\boldsymbol{s}, t_l)$ be the distribution of fields at Langevin time t_l. Then,

$$P(\boldsymbol{s}, t_l + \delta t_l) = \int D\boldsymbol{s}' P(\boldsymbol{s}', t_l) \langle \delta \left(\boldsymbol{s} - \left[\boldsymbol{s}' - \frac{\partial S(\boldsymbol{s}')}{\partial \boldsymbol{s}'} \delta t_l + \sqrt{2\delta t_l} \boldsymbol{\eta} \right] \right) \rangle_\eta \tag{47}$$

where δ corresponds to the $L_{trot} M_I$ dimensional Dirac δ-function. Taylor expanding up to order δt_l and averaging over the stochastic variable yields:

$$\begin{aligned} P(\boldsymbol{s}, t_l + \delta t_l) \quad &= \int D\boldsymbol{s}' P(\boldsymbol{s}', t_l) \times \\ &\left(\delta(\boldsymbol{s}' - \boldsymbol{s}) - \frac{\partial S(\boldsymbol{s}')}{\partial \boldsymbol{s}'} \frac{\partial}{\partial \boldsymbol{s}'} \delta(\boldsymbol{s}' - \boldsymbol{s}) \delta t_l + \frac{\partial}{\partial \boldsymbol{s}'} \frac{\partial}{\partial \boldsymbol{s}'} \delta(\boldsymbol{s}' - \boldsymbol{s}) \delta t_l \right) \\ &+ \mathcal{O}\left(\delta t_l^2\right). \end{aligned} \tag{48}$$

Partial integration and taking the limit of infinitesimal time steps gives the Fokker-Plank equation

$$\frac{\partial}{\partial t_l} P(\boldsymbol{s}, t_l) = \frac{\partial}{\partial \boldsymbol{s}} \left(P(\boldsymbol{s}, t_l) \frac{\partial S(\boldsymbol{s})}{\partial \boldsymbol{s}} + \frac{\partial P(\boldsymbol{s}, t_l)}{\partial \boldsymbol{s}} \right) \tag{49}$$

The stationary, $\frac{\partial}{\partial t_l} P(\boldsymbol{s}, t_l) = 0$, normalizable, solution to the above equation corresponds to the desired probability distribution:

$$P(\boldsymbol{s}) = \frac{e^{-S(\boldsymbol{s})}}{\int D\boldsymbol{s} e^{-S(\boldsymbol{s})}}. \tag{50}$$

As mentioned above, Langevin dynamics will work well provided that the forces show no singularities. The great advantage of such an updating scheme is that there is no rejection and that all fields are updated at each step. The following points that highlight potential issues with Langevin dynamics are in order.

- Langevin dynamics will be carried out at a finite Langevin time step and thereby we have introduced a further source of systematic error.
- The factor $\sqrt{2\delta t_l}$ multiplying the stochastic variable makes the noise dominant on short time scales. On these times scales Langevin dynamics essentially corresponds to a random walk. This has the advantage that one can circumvent potential barriers, but may render the updating scheme less efficient than the hybrid molecular dynamics approach.

In the code, we have adopted a variable time step strategy. This is important since the determinant has zeros and the forces are unbounded. The user provides an upper bound to the force, `Max_Force`, and if the maximal force in a configuration `Max_Force_Conf`, is large than `Max_Force` the time step is rescaled as

$$\tilde{\delta t_l} = \frac{\texttt{Max\_Force} * \delta t_l}{\texttt{Max\_Force\_Conf}}. \tag{51}$$

With the adaptive time step, averages are computed as:

$$\langle \hat{O} \rangle = \frac{\sum_n (\delta t_l)_n \langle \langle \hat{O} \rangle \rangle_{(C_n)}}{\sum_n (\delta t_l)_n} \tag{52}$$

We have tested the code for a 6-site Hubbard chain at half-filling at $U/t = 4$, $\beta t = 4$ and with periodic boundary conditions. One can show that for this choice of boundary conditions the forces are not bounded and to make sure that the program does not crash we have set `Max_Force = 1.5`

The discrete variable code gives

$$\langle \hat{H} \rangle = -3.468429 \pm 0.000726. \tag{53}$$

The Langevin code at $\delta t_l = 0.001$ yields

$$\langle \hat{H} \rangle = -3.456968 \pm 0.009886 \tag{54}$$

and at $\delta t_l = 0.01$

$$\langle \hat{H} \rangle = -3.495365 \pm 0.007281 \tag{55}$$

At $\delta t_l = 0.001$ the maximal force that occurred during the run was 112, whereas at $\delta t_l = 0.01$ it grew to 524. In both cases the average force was given by 0.45. For larger values of δt_l the maximal force grows and the fluctuations on the energy become larger. ($\langle \hat{H} \rangle = -3.718439 \pm 0.206469$ at $\delta t_l = 0.02$. For this parameter set the maximal force we encountered during the run was of 1658.)

Controlling Langevin dynamics when the action has logarithmic divergences is a challenge, and it is not clear that the results will be satisfying. For our specific problem we can solve this issue by considering open boundary conditions. Following an argument put forward in [48], we can show, using world lines, that the determinant is always positive. In this case the action does not have logarithmic divergences and the Langevin dynamics works beautifully well, see Fig. 1.

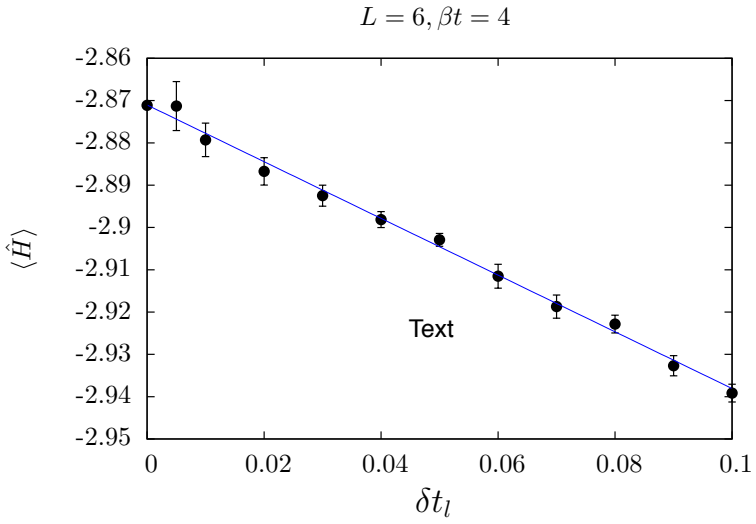

Figure 1: Total energy for the 6-sites Hubbard chain at $U/t = 4$, $\beta t = 4$ and with open boundary conditions. Here one can show that the determinant is always positive such that no singularities occur in the action, and consequently the Langevin dynamics works very well. The data point at $\delta t_l = 0$ stems from running the discrete field code with coupling of the field to the z-component of the magnetization. The extrapolated value of the energy reads -2.87104 ± 0.001144 and the reference result from the discrete code is -2.871124 ± 0.000389. Throughout the runs the maximal force was always less than the threshold of 1.5.

2.3 Stabilization - a peculiarity of the BSS algorithm

From the partition function in Eq. (15) it can be seen that, for the calculation of the Monte Carlo weight and of the observables, a long product of matrix exponentials has to be formed. In addition to that, we need to be able to extract the single-particle Green function for a given flavor index at, say, time slice $\tau = 0$. As mentioned above (cf. Eq. (19)), this quantity is given by:

$$\boldsymbol{G} = \left(\mathbb{1} + \prod_{\tau=1}^{L_{\mathrm{Trotter}}} \boldsymbol{B}_\tau \right)^{-1}, \tag{56}$$

which can be recast as the more familiar linear algebra problem of finding a solution for the linear system

$$\left(\mathbb{1} + \prod_{\tau} \boldsymbol{B}_\tau \right) x = b. \tag{57}$$

The matrices $\boldsymbol{B}_\tau \in \mathbb{C}^{n\times n}$ depend on the lattice size as well as other physical parameters that can be chosen such that a matrix norm of $\boldsymbol{B}_\tau$ can be unbound in size. From standard perturbation theory for linear systems, the computed solution $\tilde{x}$ would contain a relative error

$$\frac{|\tilde{x}-x|}{|x|} = \mathcal{O}\left(\epsilon\kappa_p\left(\mathbb{1}+\prod_\tau \boldsymbol{B}_\tau\right)\right), \tag{58}$$

where ϵ denotes the machine precision, which is 2^{-53} for IEEE double-precision numbers, and $\kappa_p(\boldsymbol{M})$ is the condition number of the matrix $\boldsymbol{M}$ with respect to the matrix p-norm. Due to $\prod_\tau \boldsymbol{B}_\tau$ containing exponentially large and small scales, as can be seen in Eq. (15), a straightforward inversion turns out to be completely ill-suited. That would lead the condition number, as a function of increasing inverse temperature, to grow exponentially, rendering the computed solution $\tilde{x}$ meaningless.

In order to circumvent this, more sophisticated methods have to be employed. As a first step, assuming that the multiplication of `NWrap` $\boldsymbol{B}$ matrices has an acceptable condition number and, for simplicity, that `NWrap` is a divisor of L_{Trotter}, we can write: (JP: I made $i \in (1, L_{\text{Trotter}}/\texttt{NWrap})$, instead of $i \in (0, L_{\text{Trotter}}/(\texttt{NWrap}-1))$ and put $i-1$ instead of i in $\boldsymbol{B}$)

$$\boldsymbol{G} = \left(\mathbb{1} + \prod_{i=1}^{\frac{L_{\text{Trotter}}}{\texttt{NWrap}}} \underbrace{\prod_{\tau=1}^{\texttt{NWrap}} \boldsymbol{B}_{(i-1)\cdot\texttt{NWrap}+\tau}}_{\equiv\mathcal{B}_i}\right)^{-1}. \tag{59}$$

The default stabilization strategy in the auxiliary-field QMC implementation of the ALF project, is then to form a product of QR-decompositions, which was proven to be weakly backwards stable in [65]. The key idea is to efficiently separate the scales of a matrix from the orthogonal part of a matrix. This can be achieved using a QR decomposition of a matrix $\boldsymbol{A}$ in the form $\boldsymbol{A}_i = \boldsymbol{Q}_i\boldsymbol{R}_i$. The matrix $\boldsymbol{Q}_i$ is unitary and hence in the usual 2-norm it holds that $\kappa_2(\boldsymbol{Q}_i) = 1$. To get a handle on the condition number of $\boldsymbol{R}_i$ we will form the diagonal matrix

$$(\boldsymbol{D}_i)_{n,n} = |(\boldsymbol{R}_i)_{n,n}| \tag{60}$$

and set $\tilde{\boldsymbol{R}}_i = \boldsymbol{D}_i^{-1}\boldsymbol{R}_i$ This gives the decomposition

$$\boldsymbol{A}_i = \boldsymbol{Q}_i\boldsymbol{D}_i\tilde{\boldsymbol{R}}_i. \tag{61}$$

$\boldsymbol{D}_i$ now contains the row norms of the original $\boldsymbol{R}_i$ matrix and hence attempts to separate off the total scales of the problem from $\boldsymbol{R}_i$. This is similar in spirit to the so-called matrix equilibration which tries to improve the condition number of a matrix through suitably chosen column and row scalings. Due to a theorem by van der Sluis [66] we know that the choice in Eq. (60) is almost optimal among all diagonal matrices $\boldsymbol{D}$ from the space of diagonal matrices $\mathcal{D}$, in the sense that

$$\kappa_p((\boldsymbol{D}_i)^{-1}\boldsymbol{R}_i) \leq n^{1/p}\min_{\boldsymbol{D}\in\mathcal{D}}\kappa_p(\boldsymbol{D}^{-1}\boldsymbol{R}_i).$$

Now, given an initial decomposition of $\boldsymbol{A}_{j-1} = \prod_i \mathcal{B}_i = \boldsymbol{Q}_{j-1}\boldsymbol{D}_{j-1}\boldsymbol{T}_{j-1}$ an update $\mathcal{B}_j\boldsymbol{A}_{j-1}$ is formed in the following three steps:

1. Form $\boldsymbol{M}_j = (\mathcal{B}_j\boldsymbol{Q}_{j-1})\boldsymbol{D}_{j-1}$. Note the parentheses.
2. Do a QR decomposition of $\boldsymbol{M}_j = \boldsymbol{Q}_j\boldsymbol{D}_j\boldsymbol{R}_j$. This gives the final $\boldsymbol{Q}_j$ and $\boldsymbol{D}_j$.
3. Form the updated $\boldsymbol{T}$ matrices $\boldsymbol{T}_j = \boldsymbol{R}_j\boldsymbol{T}_{j-1}$.

The effectiveness of the stabilization *has* to be judged for every simulation from the output file `info` (Sec. 3.3.3). For most simulations there are two values to look out for:

- `Precision Green`
- `Precision Phase`

The Green function, as well as the average phase, are usually numbers with a magnitude of $\mathcal{O}(1)$. For that reason we recommend that `NWrap` is chosen such that the mean precision is of the order of 10^{-8} or better. We include typical values of `Precision Phase` and of the mean and the maximal values of `Precision Green` in the example simulations discussed in Sec. 5.1.3 and Sec. 5.2.3.

2.4 Monte Carlo sampling

Error estimates in Monte Carlo simulations can be delicate and are based on the central limit theorem [67]. This theorem requires independent measurements and a finite variance. In this subsection we will give examples of the issues that a user will have to look out for while using a Monte Carlo code. Those effects are part of the common lore of the field and we can only cover them briefly in this text. For a deeper understanding of the inherent issues of Markov-chain Monte Carlo methods we refer the reader to the pedagogical introduction in chapter 1.3.5 of Krauth[68], the overview article of Sokal [41], the more specialized literature by Geyer [69] and chapter 6.3 of Neal [70].

In general, one distinguishes local from global updates. As the name suggest, the local update corresponds to a small change of the configuration, e.g., a single spin flip of one of the $L_{\mathrm{Trotter}}(M_I + M_V)$ field entries (see Sec. 2.2), whereas a global update changes a significant part of the configuration. The default update scheme of the ALF implementation are local updates, such that there is a minimum number of moves required for generating an independent configuration. The associated time scale is called the autocorrelation time, T_{auto}, and is generically dependent upon the choice of the observables.

Our unit of *sweeps* is defined such that each field is visited twice in a sequential propagation from $\tau = 0$ to $\tau = L_{\mathrm{Trotter}}$ and back. A single sweep will generically not suffice to produce an independent configuration. In fact, the autocorrelation time T_{auto} characterizes the required time scale to generate independent values of $\langle\langle \hat{O} \rangle\rangle_C$ for the observable O. This has several consequences for the Monte Carlo simulation:

- First of all, we start from a randomly chosen field configuration, such that one has to invest a time of *at least* one T_{auto}, but typically many more, in order to generate relevant, equilibrated configurations before reliable measurements are possible. This phase of the simulation is known as the warm-up or burn-in phase. In order to keep the code as flexible as possible (as different simulations might have different autocorrelation times), measurements are taken from the very beginning and, in the analysis phase, the parameter `n_skip` controls the number of initial bins that are ignored.

- Second, our implementation averages over bins with `NSWEEPS` measurements before storing the results on disk. The error analysis requires statistically independent bins in order to generate reliable confidence estimates. If bins are instead too small (averaged over a period shorter then T_{auto}), then the error bars are typically underestimated. Most of the time, however, the autocorrelation time is unknown before the simulation is started and, sometimes, single runs long enough to generate appropriately sized bins are not feasible. For this reason, we provide a rebinning facility controlled by the parameter `N_rebin` that specifies the number of bins recombined into each new bin during the error analysis. One can test the suitability of a given bin size by verifying whether a increase in size changes the error estimate (For an explicit example, see Sec. 2.4.2 and the appendix of Ref. [51]).

  The `N_rebin` variable can be used to control a further issue. The distribution of the Monte Carlo estimates $\langle\langle \hat{O} \rangle\rangle_C$ is unknown, while a result in the form (mean $\pm$ error) assumes a Gaussian distribution. Every distribution with a finite variance turns into a Gaussian one once it is folded often enough (central limit theorem). Due to the internal averaging (folding) within one bin, many observables are already quite Gaussian. Otherwise one can increase `N_rebin` further, even if the bins are already independent [71].

- The third issue concerns time-displaced correlation functions. Even if the configurations are independent, the fields within the configuration are still correlated. Hence, the data for $S_{\alpha,\beta}(\vec{k}, \tau)$ (see Sec. 3.2; Eq. (88)) and $S_{\alpha,\beta}(\vec{k}, \tau + \Delta\tau)$ are also correlated. Setting the switch `N_Cov=1` triggers the calculation of the covariance matrix in addition to the usual error analysis. The covariance is defined by

$$COV_{\tau\tau'} = \frac{1}{N_{\mathrm{Bin}}} \left\langle \left(S_{\alpha,\beta}(\vec{k}, \tau) - \langle S_{\alpha,\beta}(\vec{k}, \tau) \rangle \right) \left(S_{\alpha,\beta}(\vec{k}, \tau') - \langle S_{\alpha,\beta}(\vec{k}, \tau') \rangle \right) \right\rangle . \tag{62}$$

  An example where this information is necessary is the calculation of mass gaps extracted by fitting the tail of the time-displaced correlation function. Omitting the covariance matrix will underestimate the error.

2.4.1 The Jackknife resampling method

For each observable $\hat{A}, \hat{B}, \hat{C} \cdots$ the Monte Carlo program computes a data set of N_{Bin} (ideally) independent values where for each observable the measurements belong to the same statistical distribution.

In the general case, we would like to evaluate a function of expectation values, $f(\langle\hat{A}\rangle, \langle\hat{B}\rangle, \langle\hat{C}\rangle \cdots)$ – see for example the expression (24) for the observable including reweighting – and are interested in the statistical estimates of its mean value and the standard error of the mean. A numerical method for the statistical analysis of a given function f which properly handles error propagation and correlations among the observables is the Jackknife method, which is, like the related Bootstrap method, a resampling scheme [72]. Here we briefly review the *delete-1 Jackknife* scheme which is based on the idea to generate N_{bin} new data sets of size $N_{\text{bin}} - 1$ by consecutively removing one data value from the original set. By $A_{(i)}$ we denote the arithmetic mean for the observable $\hat{A}$, without the i-th data value A_i, namely

$$A_{(i)} \equiv \frac{1}{N_{\text{Bin}} - 1} \sum_{k=1,\, k\neq i}^{N_{\text{Bin}}} A_k \; . \tag{63}$$

As the corresponding quantity for the function $f(\langle\hat{A}\rangle, \langle\hat{B}\rangle, \langle\hat{C}\rangle \cdots)$, we define

$$f_{(i)}(\langle\hat{A}\rangle, \langle\hat{B}\rangle, \langle\hat{C}\rangle \cdots) \equiv f(A_{(i)}, B_{(i)}, C_{(i)} \cdots) \; . \tag{64}$$

Following the convention in the literature, we will denote the final Jackknife estimate of the mean by $f_{(\cdot)}$ and its standard error by Δf. The Jackknife mean is given by

$$f_{(\cdot)}(\langle\hat{A}\rangle, \langle\hat{B}\rangle, \langle\hat{C}\rangle \cdots) = \frac{1}{N_{\text{Bin}}} \sum_{i=1}^{N_{\text{Bin}}} f_{(i)}(\langle\hat{A}\rangle, \langle\hat{B}\rangle, \langle\hat{C}\rangle \cdots) \; , \tag{65}$$

and the standard error, including bias correction, is given by

$$(\Delta f)^2 = \frac{N_{\text{Bin}} - 1}{N_{\text{Bin}}} \sum_{i=1}^{N_{\text{Bin}}} \left[f_{(i)}(\langle\hat{A}\rangle, \langle\hat{B}\rangle, \langle\hat{C}\rangle \cdots) - f_{(\cdot)}(\langle\hat{A}\rangle, \langle\hat{B}\rangle, \langle\hat{C}\rangle \cdots) \right]^2 \; . \tag{66}$$

In case of $f = \langle\hat{A}\rangle$, the results (65) and (66) reduce to the plain sample average and the standard, bias corrected, estimate of the error.

2.4.2 An explicit example of error estimation

In the following we use one of our examples, the Hubbard model on a square lattice in the M_z Hubbard-Stratonovich decoupling (see Sec. 5.2), to show explicitly how to estimate errors. We will equally show that the autocorrelation time is dependent upon the choice of the observable. In fact, different observables within the same run can have different autocorrelation times and of course, this time scale depends on the parameter choice. Hence, the user has to check autocorrelations of individual observables for each simulation! Typical regions of the phase diagram that require special attention are critical points where length scales diverge.

To determine the autocorrelation time, we calculate the correlation function

$$Auto_{\hat{O}}(t_{\text{QMC}}) = \sum_{i=0}^{N_{\text{Bin}} - t_{\text{QMC}}} \frac{\left(O_i - \left\langle\hat{O}\right\rangle\right)\left(O_{i+t_{\text{QMC}}} - \left\langle\hat{O}\right\rangle\right)}{\left(O_i - \left\langle\hat{O}\right\rangle\right)\left(O_i - \left\langle\hat{O}\right\rangle\right)} \, , \tag{67}$$

where O_i refers to the Monte Carlo estimate of the observable $\hat{O}$ in the i^{th} bin. This function typically shows an exponential decay and the decay rate defines the autocorrelation time. Figure 2 (a) shows the autocorrelation functions $Auto_{\hat{O}}(t_{\text{QMC}})$ for three spin-spin-correlation functions [Eq. (88)] at momentum $\vec{k} = (\pi, \pi)$ and at $\tau = 0$:

$\hat{O} = S_{\hat{S}^z}$ for the z spin direction, $\hat{O} = (S_{\hat{S}^x} + S_{\hat{S}^y})/2$ for the xy plane, and $\hat{O} = (S_{\hat{S}^x} + S_{\hat{S}^y} + S_{\hat{S}^z})/3$ for the total spin. The Hubbard model has a $SU(2)$ spin symmetry. However, we chose a HS field which couples to the z-component of the magnetization, M_z, such that each configuration breaks this symmetry. Of course, after Monte Carlo averaging one expects restoration of the symmetry. The model, on bipartite lattices, shows spontaneous spin-symmetry breaking at $T = 0$ and in the thermodynamic limit. At finite temperatures, and within the so-called renormalized classical regime, quantum antiferromagnets have a length scale that diverges exponentially with decreasing temperatures [73]. The parameter set chosen for Fig. 2 is non-trivial in the sense that it places the Hubbard model in this renormalized classical regime where the correlation length is substantial. Figure 2 clearly shows a very short autocorrelation time for

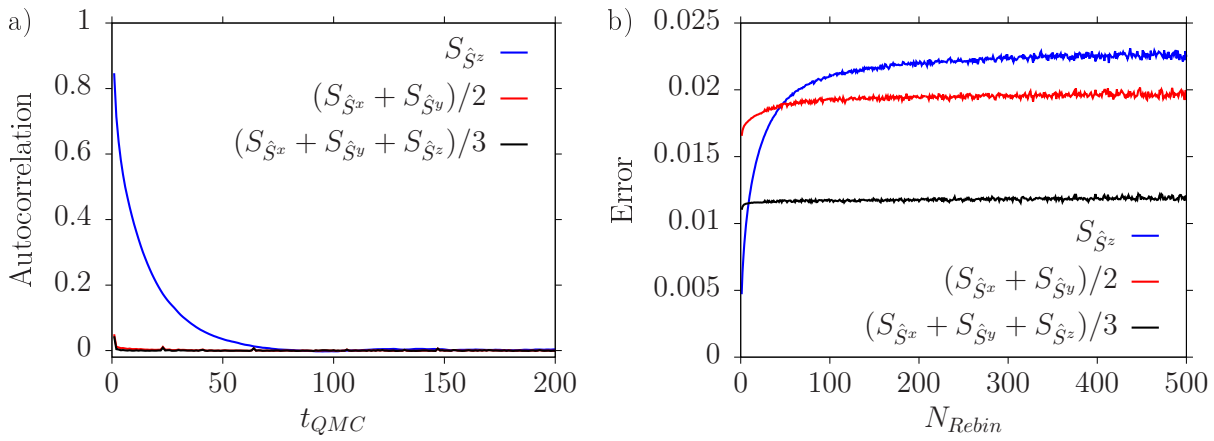

Figure 2: The autocorrelation function $Auto_{\hat{O}}(t_{\mathrm{QMC}})$ (a) and the scaling of the error with effective bin size (b) of three equal time spin-spin correlation functions $\hat{O}$ of the Hubbard model in the M_z decoupling (see Sec. 5.2). Simulations were done on a 6×6 square lattice, with $U/t = 4$ and $\beta t = 6$. The original bin contained only one sweep and we calculated around one million bins on a single core. The different autocorrelation times for the xy-plane compared to the z-direction can be detected from the decay rate of the autocorrelation function (a) and from the point where saturation of the error sets in (b), which defines the required effective bin size for independent measurements. Apparently and as argued in the text, the improved estimator $(S_{\hat{S}^x} + S_{\hat{S}^y} + S_{\hat{S}^z})/3$ has the smallest autocorrelation time.

the xy-plane whereas we detect a considerably longer autocorrelation time for the z-direction. This is a direct consequence of the *long* magnetic length scale and the chosen decoupling. The physical reason for the long autocorrelation time corresponds to the restoration of the $SU(2)$ spin symmetry. This insight can be used to define an improved, $SU(2)$ symmetric estimator for the spin-spin correlation function, namely $(S_{\hat{S}^x} + S_{\hat{S}^y} + S_{\hat{S}^z})/3$. Thereby, global spin rotations are no longer an issue and this improved estimator shows the shortest autocorrelation time as seen clearly in Fig. 2 (b). Other ways to tackle large autocorrelation can be global updates or parallel tempering.

Using the time series of Monte Carlo samples we would like to obtain estimates of the mean and the standard error of the mean. A simple method which we will describe in this tutorial is the rebinning method, also known in the literature as rebatching, where a fixed number (denoted by `N_rebin`) of adjacent original bins are aggregated to form a new effective bin. In addition to measuring the decay rate of the autocorrelation function (67), a measure for the autocorrelation time can be also obtained by the rebinning method. For a comparison to other methods of estimating the autocorrelation time we refer the reader to the literature [74, 69, 70]. A reliable error analysis requires independent bins, otherwise the error is typically underestimated. This behavior is observed in Fig. 2 (b), where the effective bin size has been systematically increased by rebinning. If the effective bin size is smaller than the autocorrelation time the error will be underestimated. When the effective bin size becomes larger than the autocorrelation time converging behavior sets in and in this region the error estimate will be correct.

For the analysis of the Monte Carlo data (see Sec. 3.5), the user can provide a finite value for `N_auto` to trigger the computation of autocorrelation functions $Auto_{\hat{O}}(t_{\mathrm{QMC}})$ in the range $t_{\mathrm{QMC}} = [0, \texttt{N\_auto}]$. Since these computations are quite time consuming and require many Monte Carlo bins the default value is `N_auto=0` if unspecified. To produce Fig. 2, we set `N_auto` $= 500$ and used a total of approximately one million bins.

2.5 Pseudo code description

Basic structure of the auxiliary field QMC implementation (`Prog/main.F90`):

Set the Hamiltonian and the lattice:
Call ham_set
Read in an auxiliary-field configuration or generate it randomly:

Call confin

This loop fills the storage, needed for the first actual Monte Carlo sweep:
Do `ntau` from `ltrot` to `1`
Compute propagation matrices and store them at the stabilization points:
Call wrapul
Enddo

Loop over bins:
Do `nbc` from `1` to `nbin`
Loop over sweeps. Each sweep updates twice (sweeping upward and downward in time) the whole space-time lattice of auxiliary fields. The sweep defines the unit of Monte Carlo time:
Do `nsw` from `1` to `nsweep`
Upward sweep:
Do `ntau` from `1` to `ltrot`
Propagate the Green function from time `ntau -1` to `ntau`, and compute a new estimate (using sequential update scheme) of the Green function at `ntau`:
Call wrapgrup

Stabilization:
If `ntau` equals stabilization point in imaginary time **then**
Compute propagation matrix from previous stabilization point to `ntau`:
Call wrapur
Read from storage: propagation from `ltrot` to `ntau`
Write to storage : the just computed propagation
Recalculate the Green function at time `ntau` in a stable way:
Call cgr
Check the precision between propagated and recalculated Green functions:
Call control_precisionG
Endif

Measure the equal time observables:
If `ntau` is in the intervall `[LOBS_ST, LOBS_EN]` **then**
Call obser
Endif
Enddo

Downward sweep:
Do `ntau` from `ltrot` to `1`
Repeat the above steps (update, propagation, stabilization, equal time measurements) for the downward direction in imaginary time
Enddo

Measure the time displaced observables:
Call tau_m
Enddo (loop over sweeps)

Calculate measurement averages for current bin and write them to disk:
call pr_obs
Write auxiliary-field configuration to disk:
call confout
Enddo (loop over bins)

3 Data Structures and Input/Output

3.1 Implementation of the Hamiltonian and the lattice

The module `Hamiltonian`, contained in the file `Hamiltonian.F90`, defines the model Hamiltonian, the lattice under consideration and the desired observables (Table 2). We have collected a number of example Hamiltonians, lattices and observables in the file `Hamiltonian_Examples.F90`. The examples are described in Sec. 5. To implement a user-defined model, only the module `Hamiltonian` has to be set up. Accordingly, this documentation focusses almost entirely on this module and the subprograms it includes. The remaining parts of the code may hence be treated as a black box.

To specify the Hamiltonian, one needs an `Operator` and a `Lattice` type as well as a type for the observables. These three data structures will be described in the following sections.

Subprogram	Description	Section
`Ham_Set`	Reads in model and lattice parameters from the file `parameters`. And it sets the Hamiltonian by calling `Ham_latt`, `Ham_hop`, and `Ham_V`.	
`Ham_hop`	Sets the hopping term $\hat{\mathcal{H}}_T$ by calling `Op_make` and `Op_set`.	3.1.1, 3.1.2
`Ham_V`	Sets the interaction terms $\hat{\mathcal{H}}_V$ and $\hat{\mathcal{H}}_I$ by calling `Op_make` and `Op_set`.	3.1.1, 3.1.2
`Ham_Latt`	Sets the lattice by calling `Make_Lattice`.	3.1.3
`S0`	A function which returns an update ratio for the Ising term $\hat{\mathcal{H}}_{I,0}$.	5.4.3
`Alloc_obs`	Assigns memory storage to the observables	
`Obser`	Computes the scalar observables and equal-time correlation functions.	3.2
`ObserT`	Computes time-displaced correlation functions.	3.2
`Init_obs`	Initializes the observables to zero.	
`Pr_obs`	Writes the observables to the disk by calling `Print_bin`.	

Table 2: Overview of the subprograms of the module `Hamiltonian` to define the Hamiltonian, the lattice and the observables. The highlighted subroutines have to be modified by the user.

3.1.1 The `Operator` type

The fundamental data structure in the code is the data structure `Operator`. It is implemented as a Fortran derived data type. This type is used to define the Hamiltonian (2). In general, the matrices $\mathbf{T}^{(ks)}$, $\mathbf{V}^{(ks)}$ and $\mathbf{I}^{(ks)}$ are sparse Hermitian matrices. Consider the matrix $\boldsymbol{X}$ of dimension $N_{\mathrm{dim}} \times N_{\mathrm{dim}}$, as a representative for each of the above three matrices. Let us denote with $\{z_1, \cdots, z_N\}$ a subset of N indices, for which

$$X_{x,y} \begin{cases} \neq 0 & \text{if } x, y \in \{z_1, \cdots z_N\} \\ = 0 & \text{otherwise} \end{cases} \tag{68}$$

Usually, we have $N \ll N_{\mathrm{dim}}$. We define the $N \times N_{\mathrm{dim}}$ matrices $\mathbf{P}$ as

$$P_{i,x} = \delta_{z_i,x} \ , \tag{69}$$

where $i \in [1, \cdots, N]$ and $x \in [1, \cdots, N_{\mathrm{dim}}]$. The matrix $\boldsymbol{P}$ selects the non-vanishing entries of $\boldsymbol{X}$, which are contained in the rank-N matrix $\boldsymbol{O}$:

$$\boldsymbol{X} = \boldsymbol{P}^T \boldsymbol{O} \boldsymbol{P} \ , \tag{70}$$

and

$$X_{x,y} = \sum_{i,j}^{N} P_{i,x} O_{i,j} P_{j,y} = \sum_{i,j}^{N} \delta_{z_i,x} O_{ij} \delta_{z_j,y} \ . \tag{71}$$

Since the $\boldsymbol{P}$ matrices have only one non-vanishing entry per column, they can conveniently be stored as a vector $\vec{P}$, with entries

$$P_i = z_i. \tag{72}$$

There are many useful identities which emerge from this structure. For example:

$$e^{\boldsymbol{X}} = e^{\boldsymbol{P}^T \boldsymbol{O} \boldsymbol{P}} = \sum_{n=0}^{\infty} \frac{\left(\boldsymbol{P}^T \boldsymbol{O} \boldsymbol{P}\right)^n}{n!} = \mathbb{1} + \boldsymbol{P}^T \left(e^{\boldsymbol{O}} - \mathbb{1}\right) \boldsymbol{P} \,, \tag{73}$$

since

$$\boldsymbol{P}\boldsymbol{P}^T = \mathbb{1}_{N \times N}. \tag{74}$$

In the code, we define a structure called `Operator` to capture the above. This type `Operator` bundles several components, listed in Table 3, that are needed to define and use an operator matrix in the program.

3.1.2 Specification of the model

Variable	Type	Description
`Op_X%N`	Integer	Effective dimension N
`Op_X%O`	Complex	Matrix $\mathbf{O}$ of dimension $N \times N$
`Op_X%P`	Integer	Matrix $\mathbf{P}$ encoded as a vector of dimension N
`Op_X%g`	Complex	Coupling strength g
`Op_X%alpha`	Complex	Constant α
`Op_X%type`	Integer	Sets the type of HS transformation (1: Ising; 2: discrete HS for perfect-square term; 3: continuous real field.)
`Op_X%diag`	Logical	True if $\mathbf{O}$ is diagonal
`Op_X%U`	Complex	Matrix containing the eigenvectors of $\mathbf{O}$
`Op_X%E`	Real	Eigenvalues of $\mathbf{O}$
`Op_X%N_non_zero`	Integer	Number of non-vanishing eigenvalues of $\mathbf{O}$
`Op_X%M_exp`	Complex	Stores `M_exp`$(:,:,s) = e^{g\phi(s,\texttt{type})\mathbf{O}(:,:)}$
`Op_X%E_exp`	Complex	Stores `E_exp`$(:,s) = e^{g\phi(s,\texttt{type})\mathbf{E}(:)}$

Table 3: Member variables of the `Operator` type. In the left column, the letter `X` is a placeholder for the letters `T` and `V`, indicating hopping and interaction operators, respectively. The highlighted variables have to be specified by the user. `M_exp` and `E_exp` are allocated only if `type` $= 1, 2$.

In this section we show how to specify the Hamiltonian (2) in the code. More precisely, we have to set the matrix representation of the imaginary-time propagators – $e^{-\Delta\tau \boldsymbol{T}^{(ks)}}$, $e^{\sqrt{-\Delta\tau U_k}\eta_{k\tau}\boldsymbol{V}^{(ks)}}$, and $e^{-\Delta\tau s_{k\tau}\boldsymbol{I}^{(ks)}}$ – that appear in the partition function (15). For each pair of indices (k, s), these terms have the general form

$$\text{Matrix Exponential} = e^{g\,\phi(\texttt{type})\,\mathbf{X}} \,. \tag{75}$$

In case of the perfect-square term, we additionally have to set the constant α, see the definition of the operators $\hat{V}^{(k)}$ in Eq. (4). The data structures which hold all the above information are variables of the type `Operator` (see Table 3). For each pair of indices (k, s), we store the following parameters in an `Operator` variable:

- $\vec{P}$ and $\mathbf{O}$ defining the matrix $\mathbf{X}$ [see Eq. (70)]
- the constants g, α
- optionally: the type `type` of the discrete fields ϕ

In case of the Ising term, we store `type=1` which sets $\phi_{k\tau} = s_{k\tau}$. In case of the perfect-square term, the field results from the discrete HS transformation (10) and we store `type=2` which sets $\phi_{k\tau} = \eta_{k\tau}$. For continuous real fields f we store `type=3` which sets $\phi_{k\tau}(f) = f$. Note that we have dropped the color index σ, since the implementation uses the $SU(N_{\text{col}})$ invariance of the Hamiltonian.

Accordingly, the following data structures fully describe the Hamiltonian (2):

- For the hopping Hamiltonian (3), we have to set the exponentiated hopping matrices $e^{-\Delta\tau \boldsymbol{T}^{(ks)}}$:

In this case $\mathbf{X}^{(ks)} = \mathbf{T}^{(ks)}$. Precisely, a single variable `Op_T` describes the operator matrix

$$\left(\sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}_x^\dagger T_{xy}^{(ks)} \hat{c}_y \right) , \tag{76}$$

where $k = [1, M_T]$ and $s = [1, N_{\mathrm{fl}}]$. To make contact with the general expression (75) we set $g = -\Delta\tau$ (and $\alpha = 0$). In case of the hopping matrix, the type variable `Op_T%type` is neglected by the code. All in all, the corresponding array of structure variables is `Op_T(`M_T`,`N_{fl}`)`.

- For the interaction Hamiltonian (4), which is of perfect-square type, we have to set the exponentiated matrices $e^{\sqrt{-\Delta\tau U_k}\eta_{k\tau}\boldsymbol{V}^{(ks)}}$:

  In this case, $\mathbf{X} = \mathbf{V}^{(ks)}$. A single variable `Op_V` describes the operator matrix:

  $$\left[\left(\sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}_x^\dagger V_{x,y}^{(ks)} \hat{c}_y \right) + \alpha_{ks} \right] , \tag{77}$$

  where $k = [1, M_V]$ and $s = [1, N_{\mathrm{fl}}]$. To make contact with the general expression (75) and to set the constant α, we choose $g = \sqrt{-\Delta\tau U_k}$ and $\alpha = \alpha_{ks}$. The discrete Hubbard-Stratonovich decomposition which is used for the perfect-square interaction, is selected by setting the type variable to `Op_V%type` $= 2$. All in all, the required structure variables `Op_V` are defined using the array `Op_V(`M_V`,`N_{fl}`)`.

- For the Ising interaction Hamiltonian (5), we have to set the exponentiated matrices $e^{-\Delta\tau s_{k\tau}\boldsymbol{I}^{(ks)}}$:

  In this case, $\boldsymbol{X} = \boldsymbol{I}^{(k,s)}$. A single variable `Op_V` then describes the operator matrix:

  $$\left(\sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}_x^\dagger I_{xy}^{(ks)} \hat{c}_y \right) , \tag{78}$$

  where $k = [1, M_I]$ and $s = [1, N_{\mathrm{fl}}]$. To make contact with the general expression (75), we set $g = -\Delta\tau$ (and $\alpha = 0$). The Ising interaction is specified by setting the type variable `Op_V%type=1`. All in all, the required structure variables are contained in the array `Op_V(`M_I`,`N_{fl}`)`.

- In case of a full interaction [perfect-square term (4) and Ising term (5)], we define the corresponding doubled array `Op_V(`M_V`+`M_I`,`N_{fl}`)` and set the variables separately for both ranges of the array according to the above.

3.1.3 The `Lattice` type

We have a lattice module which can generate one- and two-dimensional Bravais lattices. Note that the orbital structure of each unit cell has to be specified by the user in the Hamiltonian module. The user has to specify unit vectors $\vec{a}_1$ and $\vec{a}_2$ as well as the size of the lattice. The size is characterized by two vectors $\vec{L}_1$ and $\vec{L}_2$ and the lattice is placed on a torus (periodic boundary conditions):

$$\hat{c}_{\vec{i}+\vec{L}_1} = \hat{c}_{\vec{i}+\vec{L}_2} = \hat{c}_{\vec{i}} \tag{79}$$

The function call

```
Call Make_Lattice( L1, L2, a1,  a2, Latt )
```

will generate the lattice `Latt` of type `Lattice`. Note again that the orbital structure of the unit cell has to be provided by the user. The reciprocal lattice vectors are defined by:

$$\vec{a}_i \cdot \vec{g}_i = 2\pi\delta_{i,j}, \tag{80}$$

and the Brillouin zone corresponds to the Wigner-Seitz cell of the lattice. With $\vec{k} = \sum_i \alpha_i \vec{g}_i$, the k-space quantization follows from:

$$\begin{bmatrix} \vec{L}_1 \cdot \vec{g}_1 & \vec{L}_1 \cdot \vec{g}_2 \\ \vec{L}_2 \cdot \vec{g}_1 & \vec{L}_2 \cdot \vec{g}_2 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = 2\pi \begin{bmatrix} n \\ m \end{bmatrix} \tag{81}$$

such that

$$\vec{k} = n\vec{b}_1 + m\vec{b}_2 \text{ with } \quad \vec{b}_1 = \frac{2\pi}{(\vec{L}_1\cdot\vec{g}_1)(\vec{L}_2\cdot\vec{g}_2)-(\vec{L}_1\cdot\vec{g}_2)(\vec{L}_2\cdot\vec{g}_1)}\left[(\vec{L}_2\cdot\vec{g}_2)\vec{g}_1-(\vec{L}_2\cdot\vec{g}_1)\vec{g}_2\right] \text{ and}$$
$$\vec{b}_2 = \frac{2\pi}{(\vec{L}_1\cdot\vec{g}_1)(\vec{L}_2\cdot\vec{g}_2)-(\vec{L}_1\cdot\vec{g}_2)(\vec{L}_2\cdot\vec{g}_1)}\left[(\vec{L}_1\cdot\vec{g}_1)\vec{g}_2-(\vec{L}_1\cdot\vec{g}_2)\vec{g}_1\right] \quad (82)$$

Variable	Type	Description
Latt%a1_p, Latt%a2_p	Real	Unit vectors $\vec{a}_1$, $\vec{a}_2$
Latt%L1_p, Latt%L2_p	Real	Vectors $\vec{L}_1$, $\vec{L}_2$ that define the topology of the lattice. Tilted lattices are thereby possible to implement.
Latt%N	Integer	Number of lattice points, $N_{\text{unit cell}}$
Latt%list	Integer	Maps each lattice point $i = 1, \cdots, N_{\text{unit cell}}$ to a real space vector denoting the position of the unit cell: $\vec{R}_i$ = list(i,1) $\vec{a}_1$ + list(i,2) $\vec{a}_2 \equiv i_1\vec{a}_1 + i_2\vec{a}_2$
Latt%invlist	Integer	Invlist$(i_1, i_2) = i$
Latt%nnlist	Integer	j = nnlist(i, n_1, n_2), $n_1, n_2 \in [-1, 1]$ $\vec{R}_j = \vec{R}_i + n_1\vec{a}_1 + n_2\vec{a}_2$
Latt%imj	Integer	$\vec{R}_{imj(i,j)} = \vec{R}_i - \vec{R}_j$. $imj, i, j \in 1, \cdots, N_{\text{unit cell}}$
Latt%BZ1_p, Latt%BZ2_p	Real	Reciprocal space vectors $\vec{g}_i$ (See Eq. 80)
Latt%b1_p, Latt%b1_p	Real	k-quantization (See Eq. 82)
Latt%listk	Integer	Maps each reciprocal lattice point $k = 1, \cdots, N_{\text{unit cell}}$ to a reciprocal space vector $\vec{k}_k$ = listk(k,1)$\vec{b}_1$ + listk(k,2)$\vec{b}_2 \equiv k_1\vec{b}_1 + k_2\vec{b}_2$
Latt%invlistk	Integer	Invlistk$(k_1, k_2) = k$
Latt%b1_perp_p, Latt%b2_perp_p	Real	Orthonormal vectors to $\vec{b}_i$. For internal use.

Table 4: Components of the Lattice type for two-dimensional lattices using as example the default lattice name Latt. The highlighted variables have to be specified by the user. Other components of the Lattice are generated upon calling: Call Make_Lattice(L1, L2, a1, a2, Latt).

The Lattice module equally handles the Fourier transformation. For example the subroutine Fourier_R_to_K carries out the transformation:

$$S(\vec{k},:,:,:) = \frac{1}{N_{unit\,cell}}\sum_{\vec{i},\vec{j}} e^{-i\vec{k}\cdot(\vec{i}-\vec{j})} S(\vec{i}-\vec{j},:,:,:) \quad (83)$$

and Fourier_K_to_R the inverse Fourier transform

$$S(\vec{r},:,:,:) = \frac{1}{N_{unit\,cell}}\sum_{\vec{k}\in BZ} e^{i\vec{k}\cdot\vec{r}} S(\vec{k},:,:,:). \quad (84)$$

In the above, the unspecified dimensions of the structure factor can refer to imaginary-time and orbital indices.

3.1.4 The Unit_cell type

This type defines the unit cell.

Both the lattice and the unit cell are defined in the predefined structures module.

3.2 The observable types Obser_Vec and Obser_Latt

Our definition of the model includes observables [Eq. (24)]. We have defined two observable types: Obser_vec for an array of scalar observables such as the energy, and Obser_Latt for correlation functions that have the lattice symmetry. In the latter case, translation symmetry can be used to provide improved estimators and to reduce the size of the output. We also obtain improved estimators by taking measurements in the imaginary-time interval [LOBS_ST,LOBS_EN] (see the parameter file in Sec. 3.3.1) thereby

Variable	Type	Description
Norb	Integer	Number of orbitals
N_coord	Integer	Coordination number
Orb_pos(1..Norb,2)	Real	Positions of the orbitals as measured from the lattice site.

Table 5: Components of an instance Latt_unit of the Unit_cell type. The highlighted variables have to be specified by the user.

exploiting the invariance under translation in imaginary-time. Note that the translation symmetries in space and in time are *broken* for a given configuration C but restored by the Monte Carlo sampling. In general, the user defines size and number of bins in the parameter file, each bin having a given amount of sweeps. Within a sweep we run sequentially through the HS and Ising fields, from time slice 1 to time slice L_{Trotter} and back. The results of each bin are written to a file and analyzed at the end of the run.

To accomplish the reweighting of observables (see Sec. 2.1.3), for each configuration the measured value of an observable is multiplied by the factors ZS and ZP:

$$\texttt{ZS} = \text{sign}(C)\,, \tag{85}$$

$$\texttt{ZP} = \frac{e^{-S(C)}}{\text{Re}\left[e^{-S(C)}\right]}\,. \tag{86}$$

They are computed from the Monte Carlo phase of a configuration,

$$\texttt{phase} = \frac{e^{-S(C)}}{\left|e^{-S(C)}\right|}\,, \tag{87}$$

which is provided by the main program. Note that each observable structure also includes the average sign [Eq. (25)].

3.2.1 Scalar observables

This data type is described in Table 6 and is useful to compute an array of scalar observables. Consider a variable Obs of type Obser_vec. At the beginning of each bin, a call to Obser_Vec_Init in the module observables_mod.F90 will set Obs%N=0, Obs%Ave_sign =0 and Obs%Obs_vec(:)=0. Each time the main program calls the routine Obser in the Hamiltonian module, the counter Obs%N is incremented by one, the sign (see Eq. 23) is accumulated in the variable Obs%Ave_sign, and the desired observables (multiplied by the sign and $\frac{e^{-S(C)}}{\text{Re}\left[e^{-S(C)}\right]}$, see Sec. 2.1.2) are accumulated in the vector Obs%Obs_vec. At the end of the

Variable	Type	Description	Contribution of configuration C
Obs%N	Int.	Number of measurements	
Obs%Ave_sign	Real	Cumulated sign [Eq. (25)]	$\text{sign}(C)$
Obs%Obs_vec(:)	Compl.	Cumulated vector of observables [Eq. (24)]	$\langle\langle \hat{O}(:)\rangle\rangle_C \frac{e^{-S(C)}}{\text{Re}\left[e^{-S(C)}\right]}\ \text{sign}\ (C)$
Obs%File_Vec	Char.	Name of output file	

Table 6: Components of the Obser_vec type. The table lists the data included in a variable Obs of type Obser_vec.

bin, a call to Print_bin_Vec in module observables_mod.F90 will append the result of the bin in the file File_Vec_*scal*. Note that this subroutine will automatically append the suffix *_scal* to the the filename File_Vec. This suffix is important to allow automatic analysis of the data at the end of the run.

3.2.2 Equal time and time displaced correlation functions

This data type (see Table 7) is useful so as to deal with equal time as well as imaginary-time displaced correlation functions of the form:

$$S_{\alpha,\beta}(\vec{k},\tau) = \frac{1}{N_{\text{unit cell}}} \sum_{\vec{i},\vec{j}} e^{-\vec{k}\cdot(\vec{i}-\vec{j})} \left(\langle \hat{O}_{\vec{i},\alpha}(\tau) \hat{O}_{\vec{j},\beta} \rangle - \langle \hat{O}_{\vec{i},\alpha} \rangle \langle \hat{O}_{\vec{j},\beta} \rangle \right). \tag{88}$$

Variable	Type	Description	Contribution of configuration C
`Obs%N`	Int.	Number of measurements	
`Obs%Ave_sign`	Real	Cumulated sign [Eq. (25)]	$\mathrm{sign}(C)$
`Obs%Obs_latt` $(\vec{i}-\vec{j},\tau,\alpha,\beta)$	Compl.	Cumul. correl. fct. [Eq. (24)]	$\langle\langle \hat{O}_{\vec{i},\alpha}(\tau)\hat{O}_{\vec{j},\beta}\rangle\rangle_C \frac{e^{-S(C)}}{\mathrm{Re}\left[e^{-S(C)}\right]}\mathrm{sign}(C)$
`Obs%Obs_latt0(`α`)`	Compl.	Cumul. expect. value [Eq. (24)]	$\langle\langle \hat{O}_{\vec{i},\alpha}\rangle\rangle_C \frac{e^{-S(C)}}{\mathrm{Re}\left[e^{-S(C)}\right]}$ sign (C)
`Obs%File_Latt`	Char.	Name of output file	

Table 7: Components of the `Obser_latt` type. The table lists the data included in a variable `Obs` of type `Obser_latt`.

Here, translation symmetry of the Bravais lattice is explicitly taken into account. The correlation function splits in a correlated part $S^{(\mathrm{corr})}_{\alpha,\beta}(\vec{k},\tau)$ and a background part $S^{(\mathrm{back})}_{\alpha,\beta}(\vec{k})$:

$$S^{(\mathrm{corr})}_{\alpha,\beta}(\vec{k},\tau) = \frac{1}{N_{\text{unit cell}}}\sum_{\vec{i},\vec{j}} e^{-i\vec{k}\cdot(\vec{i}-\vec{j})}\langle \hat{O}_{\vec{i},\alpha}(\tau)\hat{O}_{\vec{j},\beta}\rangle \,, \tag{89}$$

$$\begin{aligned} S^{(\mathrm{back})}_{\alpha,\beta}(\vec{k}) &= \frac{1}{N_{\text{unit cell}}}\sum_{\vec{i},\vec{j}} e^{-i\vec{k}\cdot(\vec{i}-\vec{j})}\langle \hat{O}_{\vec{i},\alpha}(\tau)\rangle\langle \hat{O}_{\vec{j},\beta}\rangle \\ &= N_{\text{unit cell}}\,\langle \hat{O}_\alpha\rangle\langle \hat{O}_\beta\rangle\,\delta(\vec{k}) \,, \end{aligned} \tag{90}$$

where translation invariance in space and time has been exploited to obtain the last line. The background part depends only on the expectation value $\langle \hat{O}_\alpha\rangle$, for which we use the following estimator

$$\langle \hat{O}_\alpha\rangle \equiv \frac{1}{N_{\text{unit cell}}}\sum_{\vec{i}}\langle \hat{O}_{\vec{i},\alpha}\rangle \,. \tag{91}$$

Consider a variable `Obs` of type `Obser_latt`. At the beginning of each bin a call to `Obser_Latt_Init` in the module `observables_mod.F90` will initialize the elements of `Obs` to zero. Each time the main program calls the `Obser` or `ObserT` routines one accumulates $\langle\langle \hat{O}_{\vec{i},\alpha}(\tau)\hat{O}_{\vec{j},\beta}\rangle\rangle_C \frac{e^{-S(C)}}{\mathrm{Re}\left[e^{-S(C)}\right]}\mathrm{sign}(C)$ in `Obs%Obs_latt(`$\vec{i}-\vec{j},\tau,\alpha,\beta$`)` and $\langle\langle \hat{O}_{\vec{i},\alpha}\rangle\rangle_C \frac{e^{-S(C)}}{\mathrm{Re}\left[e^{-S(C)}\right]}$ sign (C) in `Obs%Obs_latt0(`α`)`. At the end of each bin, a call to `Print_bin_Latt` in the module `observables_mod.F90` will append the result of the bin in the specified file `Obs%File_Latt`. Note that the routine `Print_bin_Latt` carries out the Fourier transformation and prints the results in k-space. We have adopted the following naming conventions. For equal time observables, defined by having the second dimension of the array `Obs%Obs_latt(`$\vec{i}-\vec{j},\tau,\alpha,\beta$`)` set to unity, the routine `Print_bin_Latt` attaches the suffix *_eq* to `Obs%File_Latt`. For time displaced correlation functions we use the suffix *_tau*.

3.3 File structure

Directory	Description
`Prog/`	Main program and subroutines
`Libraries/`	Collection of mathematical routines
`Analysis/`	Routines for error analysis
`Examples/`	Example simulations for Hubbard-type models
`Start/`	Parameter files and scripts
`Documentation/`	Documentation of the QMC code.

Table 8: Overview of the directories.

The code package consists of the program directories `Prog/`, `Libraries/` and `Analysis/`. The example simulations corresponding to the walkthroughs of Sec. 5.1 - 5.4 are included in `Examples/`. The package content is summarized in Table 8.

3.3.1 Input files

File	Description
`parameters`	Sets the parameters for lattice, model, QMC process, and the error analysis.
`seeds`	List of integer numbers to initialize the random number generator and to start a simulation from scratch.

Table 9: Overview of the input files in `Start/` required for a simulation.

The input files are listed in Table 9. The parameter file `Start/parameters` has the following form – using as an example the $SU(2)$-symmetric Hubbard model on a square lattice (see Sec. 5.1 for a detailed walkthrough):

```
=========================================================================================
!  Variables for the Hubb program
!-----------------------------------------------------------------------------------------
&VAR_lattice
L1 = 4                     ! Length in direction a_1
L2 = 4                     ! Length in direction a_2
Lattice_type = "Square"    ! a_1 = (1,0),  a_2=(0,1),  Norb=1, N_coord=2
!Lattice_type ="Honeycomb"! a_1 = (1,0),  a_2 =(1/2,sqrt(3)/2), Norb=2, N_coord=3
Model = "Hubbard_SU2"      ! Sets Nf=1, N_sun=2. HS field couples to the density
!Model = "Hubbard_Mz"      ! Sets Nf=2, N_sun=1. HS field couples to the
                           ! z-component of magnetization.
!Model="Hubbard_SU2_Ising"! Sets Nf_1, N_sun=2 and runs only for the square lattice
                           ! Hubbard model  coupled to transverse Ising field
/
&VAR_Hubbard               ! Variables for the Hubbard model
ham_T   = 1.D0             ! Hopping parameter
ham_chem= 0.D0             ! chemical potential
ham_U   = 4.D0             ! Hubbard interaction
Beta    = 5.D0             ! inverse temperature
dtau    = 0.1D0            ! Thereby Ltrot=Beta/dtau
/

&VAR_Ising                 ! Model parameters for the Ising code
Ham_xi = 1.d0              ! Only needed if Model="Hubbard_SU2_Ising"
Ham_J  = 0.2d0
Ham_h  = 2.d0
/

&VAR_QMC                   ! Variables for the QMC run
Nwrap   = 10               ! Stabilization. Green functions will be computed from
                           ! scratch after each time interval  Nwrap*Dtau
NSweep  = 500              ! Number of sweeps
NBin    = 2                ! Number of bins
Ltau    = 1                ! 1 for calculation of time displaced Green functions;
                           ! 0 otherwise
LOBS_ST = 1                ! Start measurements at time slice LOBS_ST
LOBS_EN = 50               ! End   measurements at time slice LOBS_EN
CPU_MAX = 0.1              ! Code will stop after CPU_MAX hours.
                           ! If not specified, code will stop after Nbin bins.
/
&VAR_errors                ! Variables for analysis programs
n_skip  = 1                ! Number of bins that will be skipped.
N_rebin = 1                ! Rebinning
N_Cov   = 0                ! If set to 1 covariance will be computed
```

```
                          ! for unequal time correlation functions.
/
```

The program allows for a number of different updating schemes. If no other variables are specified in the VAR_QMC name space, then the program will run in its default mode, namely the sequential single spin-flip mode. The additional optional variables in VAR_QMC include the following:

```
&VAR_QMC                ! Variables for the QMC run
..........
Propose_S0      =.true. ! Proposes single spin flip moves  with probability exp(-S0)
Global_moves    =.true. ! Allows for global moves in space and time
N_Global        =1      ! Number of global moves  per sweep
Global_tau_moves=.true. ! Allows for global moves on a single time slice.
N_Global_tau    =10     ! Number of global moves that will be carried out on a
                        ! single time slice
Nt_sequential_start= 1  ! One can combine sequential and global moves on
                        ! a time slice.
Nt_sequential_end  =    ! The program will carry our sequential local moves
                        ! in the range  [Nt_sequential_start, Nt_sequential_end]
                        ! and then N_Global_tau  global moves
/
```

Note that if Nt_sequential_start and Nt_sequential_end are not specified and that the variable Global_tau_moves is set to true, then the program will carry out only global moves, by setting Nt_sequential_start=1 and Nt_sequential_end=0.

If the program is compiled with the parallel tempering flag, then the additional name space VAR_TEMP has to be included in the parameter file.

```
&VAR_TEMP                    ! Variables for  parallel tempering
N_exchange_steps     =6      ! Number of exchange moves (be more precise)
N_Tempering_frequency=10     ! The frequency in units of sweeps at which
                             ! the exchange moves will be carried
mpi_per_parameter_set=2      ! Number of mpi-processes per parameter set
Tempering_calc_det   =.true. ! Specifies whether the fermion weight has to
                             ! be taken into account while tempering. The
                             ! default is .true., can be set to .false. if
                             ! the parameters that get varied only enter
                             ! the Ising action S_0
/
```

3.3.2 Output: Observables

File	Description
info	After completion of the simulation, this file documents parameters of the model, the QMC run and simulation metrics (precision, acceptance rate, wallclock time).
X_scal	Results of equal time measurements of scalar observables. The placeholder X stands for the observables Kin,Pot,Part, and Ener.
Y_eq,Y_tau	Results of equal time and time displaced measurements of correlation functions. The placeholder Y stands for Green,SpinZ,SpinXY, and Den.
confout_<threadnumber>	Output files for the HS and Ising configuration.

Table 10: Overview of the standard output files. See Sec. 3.2 for the definitions of observables and correlation functions.

The standard output files are listed in Table 10. The output of the measured data is organized in bins. One bin corresponds to the arithmetic average over a fixed number of individual measurements which depends on the chosen measurement interval `[LOBS_ST,LOBS_EN]` on the imaginary-time axis and on the number `NSweep` of Monte Carlo sweeps. If the user runs an MPI parallelized version of the code, the average also extends over the number of MPI threads. The formatting of the output for a single bin depends on the observable type, `Obs_vec` or `Obs_Latt`:

- Observables of type `Obs_vec`: For each additional bin, a single new line is added to the output file. In case of an observable with `N_size` components, the formatting is

  ```
  N_size + 1    <measured value, 1> ... <measured value, N_size>    <measured sign>
  ```

  The counter variable `N_size+1` refers to the number of measurements per line, including the phase measurement. This format is required by the error analysis routine (see Sec. 3.5). Scalar observables like kinetic energy, potential energy, total energy and particle number are treated as a vector of size `N_size=1`.

- Observables of type `Obs_Latt`: For each additional bin, a new data block is added to the output file. The block consists of the expectation values [Eq. (91)] contributing to the background part [Eq. (90)] of the correlation function, and the correlated part [Eq. (89)] of the correlation function. For imaginary-time displaced correlation functions, the formatting of the block follows this scheme:

  ```
  <measured sign>  <N_orbital>  <N_unit_cell> <N_time_slices> <dtau>
  do alpha = 1, N_orbital
      ⟨Ô_α⟩
  enddo
  do i = 1, N_unit_cell
     <reciprocal lattice vector k(i)>
     do tau = 1, N_time_slices
        do alpha = 1, N_orbital
           do beta = 1, N_orbital
              ⟨ S^(corr)_{α,β}(k(i), τ)⟩
           enddo
        enddo
     enddo
  enddo
  ```

  The same block structure is used for equal time correlation functions, except for the entries `<N_time_slices>` and `<dtau>` which are not present in the latter. Using this structure for the bins as input, the full correlation function $S_{\alpha,\beta}(\vec{k},\tau)$ [Eq. (88)] is then calculated by calling the error analysis routine (see Sec. 3.5).

3.3.3 Output: Precision

The finite temperature auxiliary field QMC algorithm is known to be numerically unstable, as discussed in Sec. 2.3. The origin the numerical instabilities arises from the imaginary-time propagation which invariably leads to exponentially small and exponentially large scales. Numerical stabilization of the code is delicate and has been pioneered in Ref. [2] for the finite-temperature algorithm and in Refs. [3, 4] for the zero temperature projective algorithm. As shown in Ref. [6] scales can be omitted in the ground state algorithm – thus rendering it very stable – but have to be taken into account in the finite-temperature code. Apart from runtime information, the file `info` contains important information concerning the stability of the code. It is important to know that numerical stabilization is delicate and there is no guarantee that it will work for all models.

If the numerical stabilization turns out to be bad, one option is to reduce the value of the parameter `Nwrap` in the parameter file. For performing the stabilization of the involved matrix multiplications we rely on routines from LAPACK. Hence it is very likely that your results may change significantly if you switch the LAPACK implementation. In order to offer a simple baseline to which people can quickly switch if they want to see whether their results depend on the library used for linear algebra routines we have included parts of the LAPACK-3.6.1 reference implementation from `http://www.netlib.org/lapack/`. You can switch to the QR decomposition related routines from the LAPACK reference implementation

by including the switch `-DQRREF` into the PROGRAMMCONFIGURATION string. To use these routines you need to link against a lapack library that implements at least the LAPACK-3.4.0 interface.[4]

To provide further flexibility, we have kept the history of different stabilization schemes. Our default strategy is quick and generically works well but we have encountered some models where it fails. If this applies to your model, you can use the switch `-DSTAB2` (stabilization scheme based on the QR decomposition, but not using the LAPACK reference implementation) or `-DSTAB1` (stabilization scheme based on singular value decomposition) in the header of the file `Makefile` and recompile the code.

Typical values for the numerical precision can be found in the examples of Sec. 5 (see Sec. 5.1.3 and 5.2.3).

3.4 Scripts

Script	Description	Section
`Start/out_to_in.sh`	Copies the output configurations of HS and Ising spins to the respective input files.	3.6
`Start/analysis.sh`	Starts the error analysis.	3.5

Table 11: Overview of the bash script files.

3.5 Analysis programs

Program	Description
`cov_scal.F90`	In combination with the script `analysis.sh`, the bin files with suffix `_scal` are read in, and the corresponding files with suffix `_scalJ` are produced. They contain the result of the Jackknife rebinning analysis (see Sec. 2.4).
`cov_eq.F90`	In combination with the script `analysis.sh`, the bin files with suffix `_eq` are read in, and the corresponding files with suffix `_eqJR` and `_eqJK` are produced. They correspond to correlation functions in real and Fourier space, respectively.
`cov_tau.F90`	In combination with the script `analysis.sh`, the bin files `X_tau` are read in, and the directories `X_kx_ky` are produced for all `kx` and `ky` greater or equal to zero. Here `X` is a place holder from `Green`, `SpinXY`, etc as specified in `Alloc_obs(Ltau)` (See section 5.1.2.1). Each directory contains a file `g_kx_ky` containing the time displaced correlation function traced over the orbitals. It also contains the covariance matrix if `N_cov` is set to unity in the parameter file (see Sec. 3.3.1). Equally, a directory `X_R0` for the local time displaced correlation function is generated.
`cov_tau_ph.F90`	At compilation time the file `cov_tau_ph.F90` is generated, and should be used to compute particle-hole imaginary time correlation functions such as Spin and Charge. Here we use the fact that these correlation functions are symmetric around $\tau = \beta/2$ so that we can define an improved estimator by averaging over τ and $\beta - \tau$.

Table 12: Overview of analysis programs that are called within the script `analysis.sh`.

Here we briefly discuss the analysis programs which read in bins and carry out the error analysis. (See Sec. 2.4 for a more detailed discussion.) Error analysis is based on the central limit theorem, which requires bins to be statistically independent, and also the existence of a well-defined variance for the observable under consideration. The former will be the case if bins are longer than the autocorrelation time. The latter has to be checked by the user. In the parameter file listed in Sec. 3.3.1, the user can specify how many initial bins should be omitted (variable `n_skip`). This number should be comparable to the autocorrelation time. The rebinning variable `N_rebin` will merge `N_rebin` bins into a single new bin. If the autocorrelation time is smaller than the effective bin size, the error should become independent of the bin size and thereby of the variable `N_rebin`. Our analysis is based on the Jackknife resampling[72].

[4] We have encountered some compiling issues with this flag. In particular the older intel ifort compiler version 10.1 fails for all optimization levels.

As listed in Table 12 we provide three analysis programs to account for the three observable types. The programs can be found in the directory `Analysis` and are executed by running the bash shell script `analysis.sh`. In the following, we describe the formatting of the output files mentioned in Table 14.

File	Description
`parameters`	Contains also variables for the error analysis: `n_skip`, `N_rebin` and `N_Cov` (see Sec. 3.3.1)
`X_scal`, `Y_eq`, `Y_tau`	Monte Carlo bins (see Table 10)

Table 13: Standard input files for the error analysis.

File	Description
`X_scalJ`	Jackknife mean and error of `X`, where `X` stands for `Kin, Pot, Part`, and `Ener`.
`Y_eqJR` and `Y_eqJK`	Jackknife mean and error of `Y`, where `Y` stands for `Green, SpinZ, SpinXY`, and `Den`. The suffixes `R` and `K` refer to real and reciprocal space, respectively.
`Y_R0/g_R0`	Time-resolved and spatially local Jackknife mean and error of `Y`, where `Y` stands for `Green, SpinZ, SpinXY`, and `Den`.
`Y_kx_ky/g_kx_ky`	Time resolved and $\vec{k}$-dependent Jackknife mean and error of `Y`, where `Y` stands for `Green, SpinZ, SpinXY`, and `Den`.

Table 14: Standard output files of the error analysis.

- For the scalar quantities `X`, the output files `X_scalJ` have the following formatting:

```
Effective number of bins, and bins:          <N_bin - n_skip>          <N_bin>

OBS :    1      <mean(X)>      <error(X)>

OBS :    2      <mean(sign)>   <error(sign)>
```

- For the equal time correlation functions `Y`, the formatting of the output files `Y_eqJR` and `Y_eqJK` follows this structure:

```
do i = 1, N_unit_cell
   <k_x(i)>   <k_y(i)>
   do alpha = 1, N_orbital
   do beta  = 1, N_orbital
      alpha   beta   Re<mean(Y)>   Re<error(Y)>   Im<mean(Y)>   Im<error(Y)>
   enddo
   enddo
enddo
```

  where `Re` and `Im` refer to the real and imaginary part, respectively.

- The imaginary-time displaced correlation functions `Y` are written to the output files `Y_R0/g_R0`, when measured locally in space, and to the output files `Y_kx_ky/g_kx_ky` when they are measured $\vec{k}$-resolved. The first line of the file prints the number of time slices, the number of bins and the inverse temperature. Both output files have the following formatting:

```
do i = 0, Ltau
   tau(i)   <mean( Tr[Y] )>   <error( Tr[Y])>
enddo
```

  where `Tr` corresponds to the trace over the orbital degrees of freedom. For particle-hole quantities at finite temperature, τ runs from 0 to $\beta/2$. In all other cases it runs from 0 to β.

3.6 Running the code

In this section we describe the steps how to compile and run the code, as well as how to perform the error analysis of the data.

3.6.1 Compilation

The environment variables and the directives to compile the code are set in the following makefile `Makefile`:

```
# -DMPI selects MPI.
# -DTEMPERING selects tempering mode.  MPI has to be switched on.
# -DSTAB1   Alternative stabilization, using the singular value decomposition.
# -DSTAB2   Alternative stabilization, lapack QR with  manual pivoting.
#           Packed form of QR factorization is not used.
# -DSTAB3   Alternative stabilization, using QR  with pivoting.
#           Internally, scales larger and smaller one are distinguished.
# -DLOG     Alternative stabilization, using QR  with pivoting.
#           Internally, scales are stored on log axsis to allow larger beta and
#           larger and smaller have to be distinguished.
# (no flag) Default  stabilization, using lapack QR with pivoting.
#           Packed form of QR factorization  is used.
# -DQRREF   Enables reference lapack implementation of QR decomposition.
# Recommendation: just use the -DMPI flag if you want to run in parallel or
#                 leave it empty for serial jobs.
#                 The default stabilization, no flag, is generically the best.
#                 Consider using -DLOG if you run into overflows
PROGRAMCONFIGURATION = -DMPI
PROGRAMCONFIGURATION =
f90 = gfortran
export f90
F90OPTFLAGS = -O3 -Wconversion  -fcheck=all
F90OPTFLAGS = -O3
export F90OPTFLAGS
F90USEFULFLAGS = -cpp -std=f2003
F90USEFULFLAGS = -cpp
export F90USEFULFLAGS
FL = -c ${F90OPTFLAGS} ${PROGRAMCONFIGURATION}
export FL
DIR = ${CURDIR}
export DIR
Libs = ${DIR}/Libraries/
export Libs
LIB_BLAS_LAPACK = -llapack -lblas
export LIB_BLAS_LAPACK

all: lib ana program

lib:
        cd Libraries && $(MAKE)
ana:
        cd Analysis && $(MAKE)
program:
        cd Prog && $(MAKE)


clean: cleanall
cleanall: cleanprog cleanlib cleanana
cleanprog:
```

```
	cd Prog && $(MAKE) clean
cleanlib:
	cd Libraries && $(MAKE) clean
cleanana:
	cd Analysis && $(MAKE) clean
help:
	@echo "The following are some of the valid targets of this Makefile"
	@echo "all, program, lib, ana, clean, cleanall, cleanprog, cleanlib,
	       cleanana"
```

In the above, the GNU Fortan compiler `gfortran` is set.[5] We provide a set of options for compilation of the QMC code. The present options are `-DMPI`, `-DQRREF`, `-DSTAB1`, and `-DSTAB2`. They can be included in the string variable `PROGRAMCONFIGURATION` by the user, as shown above. The program can be compiled and ran either in single-thread mode (default) or in multi-threading mode (define `-DMPI`) using the MPI standard for parallelization. The remaining three compiler options select a particular stabilization scheme for the matrix multiplications (see Sec. 3.3.3). To compile the libraries, the analysis routines and the QMC program at once, just execute the single command:

```
make
```

To clean up all directories and remove the object files and executables, execute the command `make clean`. As can be seen in the above makefile, there exist also rules to compile/clean up the library, the analysis routines and the QMC program separately.

3.6.2 Starting a simulation

To start a simulation from scratch, the following files have to be present: `parameters` and `seeds`. To run a single-thread simulation, for example by using the parameters of one of the Hubbard models described in Sec. 5, issue the command

```
./Prog/Examples.out
```

To restart the code using an existing simulation as a starting point, first run the script `out_to_in.sh` to set the input configuration files.

3.6.3 Error analysis

Note that the error analysis script requires the presence of the environment variable `DIR` which defines the path to the error analysis programs. So before starting the error analysis, one has to make this variable available which is done by the script `setenv.sh`. The command is

```
source ./setenv.sh
```

To perform an error analysis based on the Jackknife resampling method (Sec. 2.4.1) of the Monte Carlo bins for all observables run the script `analysis.sh` (see Sec. 3.5). In case that the parameter `N_auto` is set to a finite value the script will also trigger the computation of autocorrelation functions (Sec. 2.4.2).

4 Maximum entropy

4.1 General setup

Generically, the maximum entropy code computes the image $A(\omega)$ for a given data set $g(\tau)$ and kernel $K(\tau, \omega)$:

$$g(\tau) = \int_{\omega_{\text{start}}}^{\omega_{\text{end}}} d\omega K(\tau, \omega) A(\omega). \tag{92}$$

[5] A known issue with the alternative Intel Fortran compiler `ifort` is the handling of automatic, temporary arrays which `ifort` allocates on the stack. For large system sizes and/or low temperatures this may lead to a runtime error. One solution is to demand allocation of arrays above a certain size on the heap instead of the stack. This is accomplished by the `ifort` compiler flag `-heap-arrays [n]` where `[n]` is the minimal size (in kilobytes, for example `n=1024`) of arrays that are allocated on the heap.

The ALF-package includes a standard implementation of the stochastic MaxEnt as formulated in the article of K. Beach Ref. [75]. Here we will comment on the workflow. The module `Libraries/Modules/maxent_stoch.F90` contains a general implementation and the wrapper is in `Analysis/Max_SAC.F90`.

The stochastic MaxEnt is essentially a parallel tempering Monte Carlo simulation. For a discrete set of τ_i points, $i \in 1 \cdots n$ the energy reads

$$\chi^2(A) = \sum_{i,j=1}^{n} \left[g(\tau_i) - \overline{g(\tau_i)} \right] C^{-1}(\tau_i, \tau_j) \left[g(\tau_j) - \overline{g(\tau_j)} \right] \tag{93}$$

with $\overline{g(i)} = \int d\omega K(\tau_i, \omega) A(\omega)$ and C the covariance matrix. The set of inverse temperatures we will consider in the parallel tempering reads: $\alpha_m = \alpha_{st} R^m$, $m = 1 \cdots N_\alpha$. The phase space corresponds to all possible spectral functions with given sum rule and required positivity. Finally, the partition function reads $Z = \int DA e^{-\alpha \chi^2(A)}$.

In the code, the spectral function is parametrized by a set of Dirac δ functions:

$$A(\omega) = \sum_{i=1}^{N_\gamma} a_i \delta(\omega - \omega_i) . \tag{94}$$

To produce a histogram of $A(\omega)$ we divide the frequency range in `Ndis` intervals. The Green function is read from the file `g_dat` corresponding to the output of the the `cov_tau.F90` analysis program. Below, we summarize the parameters in name-lists `VAR_Max_Stoch` and `VAR_errors` in the `parameters` file required to run the maxent code .

```
&VAR_Max_Stoch              ! Variables for Stochastic Maximum entropy
Ngamma                      ! # of  Dirac functions for parametrization
Om_st                       ! Frequency range lower bound
Om_en                       ! Frequency range upper bound
NDis                        ! # of boxes for histogram
Nbins                       ! # of bins for Monte Carlo
Nsweeps                     ! # of sweeps per bin
NWarm                       ! The Nwarm first bins will be ommitted
N_alpha                     ! # of tempertures
alpha_st                    ! smallest inverse temperature
R                           ! increment for inverse temperature (see above)
Channel                     ! T0       : Zero temperature
                            ! P        : Finite temperarure particle
                            ! PH       : Finite temperarure particle-hole
                            ! PP       : Finite temperarure particle-particle
Checkpoint                  !.true.    : dump files will be produced so
                            !            as to be able to restart the simulation
                            !.false.   : dump files will not be produced
Tolerance                   !Data points for which the relative error
                            !exceeds the tolerance threshold will be omitted.
/

&VAR_erros                  ! Variables for the error analyis
....                        !
N_cov                       ! =1  Covariance will be taken into account
                            ! =0  Covariance will not be taken into account
/
```

Note that the variable `N_cov` req

Output files

The code produces the following output files.

- The files `Aom_n` correspond to the average spectral function at inverse temperature α_n. This corresponds to $\langle A_n(\omega) \rangle = \frac{1}{Z} \int DA(\omega) e^{-\alpha_n \chi^2(A)} A(\omega)$. The file contains three colums ω, $\langle A_n(\omega) \rangle$, $\Delta \langle A_n(\omega) \rangle$.
- The files `Aom_ps_n` contain the average image over the inverse temperatures α_n to α_{N_γ} see Ref. [75] for more details. The first three columns have the same meaning as for the files `Aom_n`

- The file `Green` contains the Green function. The three columns correspond to ω, $\mathrm{Re}G(\omega)$, $\mathrm{Im}G(\omega)$. This is obtained from the spectral function through:

$$G(\omega) = -\frac{1}{\pi}\int d\Omega \frac{A(\Omega)}{\omega - \Omega + i\delta} \tag{95}$$

  where $\delta = \Delta\omega$ with $\Delta\omega = (\omega_{\rm end} - \omega_{\rm start})/\mathrm{Ndis}$ and the image corresponds to that of the file `Aom_ps_m` with $m = N_\alpha - 10$. The first column of the `Green` file is a place holder for post-processing. The last three columns correspond to $\omega, \mathrm{Re}G(\omega), -\mathrm{Im}G(\omega)/\pi$.

- One of the most important files is the file `energies`. It contains there columns: $\alpha_n, \langle\chi^2\rangle, \Delta\langle\chi^2\rangle$.

- `best_fit` gives the values of a_i and ω_i (recall that $A(\omega) = \sum_{i=1}^{N_\gamma} a_i \delta(\omega - \omega_i)$) corresponding to the last configuration of the lowest temperature run.

- The File `data_out` is a crosscheck. It plots $\tau, g(\tau), \Delta g(\tau), \int d\omega K(\tau,\omega)A(\omega)$ where the image corresponds to the best fit (i.e. the lowest temperature). This file will give you a feeling on how good the fit actually is. Note that `data_out` contains only the data points that have passed the tolerance test.

- There are two `dump` files which are generated. Since the MaxEnt is a Monte Carlo code, one would like to be able to continue a simulation to improve. The data in the dump files will allow you to pursue the simulation without loosing the first run(s). These files are only generated if the variable `checkpoint` is set to true.

The essential question is: which image should one use. There is no real answer to this question in the context of the stochastic MaxEnt. The only rule of thumb is to consider temperatures for which the χ^2 is comparable to the number of data points.

4.2 Single particle quantities

For the single-particle Green function,

$$\langle \hat{c}_k(\tau)\hat{c}_k^\dagger(0)\rangle = \int d\omega K_p(\tau,\omega) A_p(k,\omega) \tag{96}$$

with

$$K_p(\tau,\omega) = \frac{1}{\pi}\frac{e^{-\tau\omega}}{1+e^{-\beta\omega}} \tag{97}$$

and in the Lehmann representation,

$$A_p(k,\omega) = \frac{\pi}{Z}\sum_{n,m} e^{-\beta E_n}\left(1 + e^{-\beta\omega}\right)|\langle n|c_n|m\rangle|^2 \delta\left(E_m - E_n - \omega\right) \tag{98}$$

Here $\left(\hat{H} - \mu\hat{N}\right)|n\rangle = E_n|n\rangle$.

Note that $A_p(k,\omega) = -\mathrm{Im}G^{\rm ret}(k,\omega)$ with

$$G^{\rm ret}(k,\omega) = -i\int dt \Theta(t) e^{i\omega t}\langle\left\{\hat{c}_k(t), \hat{c}_k^\dagger(0)\right\}\rangle \tag{99}$$

Finally the sum rule reads:

$$\int d\omega A_p(k,\omega) = \pi\langle\left\{\hat{c}_k, \hat{c}_k^\dagger\right\}\rangle = \pi \tag{100}$$

Using the `Max_Sac.F90` with `Channel="P"` will load the above Kernel in the MaxEnt library. Note that in this case the back transformation is set to unity. Note that since for each configuration of fields, $\langle\langle\hat{c}_k(\tau=0)\hat{c}_k^\dagger(0)\rangle\rangle_C + \langle\langle\hat{c}_k(\tau=\beta)\hat{c}_k^\dagger(0)\rangle\rangle_C = \langle\langle\left\{\hat{c}_k, \hat{c}_k^\dagger\right\}\rangle\rangle_C = 1$. Hence if both the $\tau = 0$ anf $\tau = \beta$ data points are included, the covariance matrix will have a zero eigenvalue and the χ^2. measure is not defined. Hence for the particle channel, the program omits the $\tau = \beta$ data point. One should also not that there are special particle-hole symmetric cases where the $\tau = 0$ data point shows no fluctuations. In this case, the code equally omits the $\tau = 0$ data point.

4.3 Particle-hole quantities

Imaginary time formulation. For particle-hole quantities such as spin-spin or charge-charge correlations, the Kernel reads:

$$\langle \hat{S}(q,\tau)\hat{S}(-q,0)\rangle = \frac{1}{\pi}\int \mathrm{d}\omega \frac{e^{-\tau\omega}}{1-e^{-\beta\omega}}\chi''(q,\omega). \tag{101}$$

This follows directly from the Lehmann representation:

$$\chi''(q,\omega) = \frac{\pi}{Z}\sum_{n,m} e^{-\beta E_n}|\langle n|\hat{S}(q)|\rangle m|^2\delta(\omega+E_n-E_m)\left(1-e^{-\beta\omega}\right) \tag{102}$$

Since the linear response to a Hermitian perturbation is real, $\chi''(q,\omega) = -\chi''(-q,-\omega)$. Hence for systems with inversion symmetry – that we will consider here – $\langle \hat{S}(q,\tau)\hat{S}(-q,0)\rangle$ is a symmetric function around $\beta = \tau/2$. The analysis file `cov_tau_ph.F90` produced at compilation time will use this to define an improved estimator.

The Stochastic MaxEnt requires a sum rule, such that the Kernel and image have to be adequately redefined. Consider:

$$\coth(\beta\omega/2)\chi''(q,\omega) \tag{103}$$

For this quantitiy, we have the sum rule since:

$$\int \mathrm{d}\omega \coth(\beta\omega/2)\chi''(q,\omega) = 2\pi\langle \hat{S}(q,\tau=0)\hat{S}(-q,0)\rangle \tag{104}$$

which is just the first point in the data.

Hence,

$$\langle \hat{S}(q,\tau)\hat{S}(-q,0)\rangle = \int \mathrm{d}\omega \underbrace{\frac{1}{\pi}\frac{e^{-\tau\omega}}{1-e^{-\beta\omega}}\tanh(\beta\omega/2)}_{K_{pp}(\tau,\omega)} \underbrace{\coth(\beta\omega/2)\chi''(q,\omega)}_{A(\omega)} \tag{105}$$

and one computes $A(\omega)$. Note that since χ'' is an odd function of ω one restricts the integration range positive values of ω. Hence:

$$\langle \hat{S}(q,\tau)\hat{S}(-q,0)\rangle = \int_0^\infty \mathrm{d}\omega \underbrace{\left(K(\tau,\omega)+K(\tau,-\omega)\right)}_{K_{ph}(\tau,\omega)} A(\omega). \tag{106}$$

In the code, ω_{start} is set to zero by default and the Kernel K_{ph} is used in the code and is defined in the routine `XKER_ph`. In general, one would like to produce the dynamical structure factor that relates to the susceptibility according to

$$S(q,\omega) = \chi''(q,\omega)/\left(1-e^{-\beta\omega}\right). \tag{107}$$

In the code the routine `BACK_TRANS_ph` transforms the image A to the desired quantity.

$$S(q,\omega) = \frac{A(\omega)}{1+e^{-\beta\omega}} \tag{108}$$

Matsubara frequency formulation. The ALF library uses imaginary time. It is however possible to formulate the MaxEnt in Matsubara frequencies. Consider:

$$\chi(q,i\Omega_m) = \int_0^\beta \mathrm{d}\tau e^{i\Omega_m\tau}\langle S(q,\tau)S(-q,0)\rangle = \frac{1}{\pi}\int \mathrm{d}\omega \frac{\chi''(q,\omega)}{\omega - i\Omega_m}. \tag{109}$$

Using the fact that $\chi''(q,\omega) = -\chi''(-q,-\omega) = -\chi''(q,-\omega)$ one obtains:

$$\begin{aligned}\chi(q,i\Omega_m) &= \frac{1}{\pi}\int_0^\infty \mathrm{d}\omega\left(\frac{1}{\omega-i\Omega_m}-\frac{1}{-\omega-i\Omega_m}\right)\chi''(q,\omega)\\ &= \frac{2}{\pi}\int_0^\infty \mathrm{d}\omega\frac{\omega^2}{\omega^2+\Omega_m^2}\frac{\chi''(q,\omega)}{\omega} \equiv \int_0^\infty \mathrm{d}\omega K(\omega,i\Omega_m)A(q,\omega)\end{aligned} \tag{110}$$

with

$$K(\omega, i\Omega_m) = \frac{\omega^2}{\omega^2 + \Omega_m^2} \tag{111}$$

and

$$A(q,\omega) = \frac{2}{\pi}\frac{\chi''(q,\omega)}{\omega} \tag{112}$$

The above definitions are useful since the image satisfies the sum rule:

$$\int_0^\infty \mathrm{d}\omega A(q,\omega) = \frac{1}{\pi}\int_{-\infty}^{\infty} \mathrm{d}\omega \frac{\chi''(q,\omega)}{\omega} \equiv \chi(q, i\Omega_m = 0) \tag{113}$$

4.4 Particle-Particle quantities

Similarly to the particle-hole channel the particle-particle channel is also a bosonic correlation function. Here however we do not assume that the imaginary time data is symmetric around the $\tau = \beta/2$ point. We use the Kernel K_{pp} define in Eq. 105 and consider the whole frequency range. The back transformation yields

$$\frac{\chi''(\omega)}{\omega} = \frac{\tanh(\beta\omega/2)}{\omega} A(\omega) \tag{114}$$

4.5 Zero temperature, projective code

In the zero temperature limit, the spectral function associated to an operator $\hat{O}$ reads:

$$A_o(\omega) = \pi \sum_n |\langle n|\hat{O}|0\rangle|^2 \delta(E_n - E_0 - \omega) \tag{115}$$

such that

$$\langle 0|\hat{O}^\dagger(\tau)\hat{O}(0)|0\rangle = \int d\omega K_0(\tau,\omega) A_0(\omega) \tag{116}$$

with

$$K_0(\tau,\omega) = \frac{1}{\pi} e^{-\tau\omega}. \tag{117}$$

The zeroth moment of the spectral function reads,

$$\int d\omega A_o(\omega) = \pi \langle 0|\hat{O}^\dagger(0)\hat{O}(0)|0\rangle, \tag{118}$$

and hence corresponds to the first data point. In the zero-temperature limit one does not distinguish between particle, particle-hole, or particle-particle channels. Using the `Max_Sac.F90` with `Channel="T0"` will load the above Kernel in the MaxEnt library. In this case the back transformation is set to unity. The code will also cut-off the tail of the imaginary time correlation function if the relative error is greater that the variable `Tolerance`.

5 Examples

5.1 The $SU(2)$-Hubbard model on a square lattice

To implement a Hamiltonian, the user has to provide a module which specifies the lattice, the model, as well as the observables they wish to compute. In this section, we describe the module `Hamiltonian_Examples.F90` which contains an implementation of the Hubbard model on the square lattice. A sample run for this model can be found in `Examples/Hubbard_SU2_Square/`.

The Hamiltonian reads

$$\mathcal{H} = \sum_{\sigma=1}^{2} \sum_{x,y=1}^{N_{\text{unit cell}}} c_{x\sigma}^\dagger T_{x,y} c_{y\sigma} + \frac{U}{2} \sum_x \left[\sum_{\sigma=1}^{2} \left(c_{x\sigma}^\dagger c_{x\sigma} - 1/2 \right) \right]^2 . \tag{119}$$

We can make contact with the general form of the Hamiltonian by setting: $N_{\text{fl}} = 1$, $N_{\text{col}} \equiv$ `N_SUN` $= 2$, $M_T = 1$, $T_{xy}^{(ks)} = T_{x,y}$, $M_V = N_{\text{unit cell}}$, $U_k = -\frac{U}{2}$, $V_{xy}^{(ks)} = \delta_{x,y}\delta_{x,k}$, $\alpha_{ks} = -\frac{1}{2}$ and $M_I = 0$.

5.1.1 Setting the Hamiltonian: `Ham_set`

The main program will call the subroutine `Ham_set` in the module `Hamiltonian_Hub.F90`. The latter subroutine defines the public variables

```
Type (Operator), dimension(:,:), allocatable  :: Op_V
Type (Operator), dimension(:,:), allocatable  :: Op_T
Integer, allocatable :: nsigma(:,:)
Integer              :: Ndim,  N_FL,  N_SUN,  Ltrot
```

which specify the model. The array `nsigma` contains the HS field. The routine `Ham_set` will first read the parameter file, then set the lattice, `Call Ham_latt`, set the hopping, `Call Ham_hop`, and set the interaction, `call Ham_V`. The parameters are read in from the file `parameters`, see Sec. 3.3.1.

5.1.1.1 The lattice: `Call Ham_latt`

The choice `Lattice_type = "Square"` sets $\vec{a}_1 = (1,0)$ and $\vec{a}_2 = (0,1)$ and for an $L_1 \times L_2$ lattice $\vec{L}_1 = L_1\vec{a}_1$ and $\vec{L}_2 = L_2\vec{a}_2$. The call to `Call Make_Lattice( L1, L2, a1, a2, Latt)` will generate the lattice `Latt` of type `Lattice`. For the Hubbard model on the square lattice, the number of orbitals per unit cell is given by `NORB=1` such that $N_{\mathrm{dim}} \equiv N_{\text{unit cell}} \cdot$ `NORB` $=$ `Latt%N` $\cdot$ `NORB`, since $N_{\text{unit cell}} =$ `Latt%N`.

5.1.1.2 The hopping term: `Call Ham_hop`

The hopping matrix is implemented as follows. We allocate an array of dimension 1×1 of type operator called `Op_T` and set the dimension for the hopping matrix to $N = N_{\mathrm{dim}}$. One allocates and initializes this type by a single call to the subroutine `Op_make`:

```
call Op_make(Op_T(1,1),Ndim)
```

Since the hopping does not break down into small blocks, we have $\boldsymbol{P} = \mathbb{1}$ and

```
Do i= 1,Ndim
  Op_T(1,1)%P(i) = i
Enddo
```

We set the hopping matrix with

```
DO I = 1, Latt%N
   Ix = Latt%nnlist(I,1,0)
   Iy = Latt%nnlist(I,0,1)
   Op_T(1,1)%O(I  ,Ix) = cmplx(-Ham_T,   0.d0,kind(0.D0))
   Op_T(1,1)%O(Ix,I  ) = cmplx(-Ham_T,   0.d0,kind(0.D0))
   Op_T(1,1)%O(I  ,Iy) = cmplx(-Ham_T,   0.d0,kind(0.D0))
   Op_T(1,1)%O(Iy, I ) = cmplx(-Ham_T,   0.d0,kind(0.D0))
   Op_T(1,1)%O(I  ,I ) = cmplx(-Ham_chem,0.d0,kind(0.D0))
ENDDO
```

Here, the integer function `j= Latt%nnlist(I,n,m)` is defined in the lattice module and returns the index of the lattice site $\vec{I} + n\vec{a}_1 + m\vec{a}_2$. Note that periodic boundary conditions are already taken into account. The hopping parameter `Ham_T` as well as the chemical potential `Ham_chem` are read from the parameter file. To completely define the hopping we further set: `Op_T(1,1)%g = -Dtau` , `Op_T(1,1)%alpha = cmplx(0.d0,0.d0, kind(0.D0))` and call the routine `Op_set(Op_T(1,1))` so as to generate the unitary transformation and eigenvalues as specified in Table 3. Recall that for the hopping, the variable `Op_set(Op_T(1,1))%type` is not required. Note that although a checkerboard decomposition is not used here, it can be implemented by considering a larger number of sparse hopping matrices.

5.1.1.3 The interaction term: `Call Ham_V`

To implement this interaction, we allocate an array of `Operator` type. The array is called `Op_V` and has dimensions $N_{\rm dim} \times N_{\rm fl} = N_{\rm dim} \times 1$. We set the dimension for the interaction term to $N = 1$, and allocate and initialize this array of type `Operator` by repeatedly calling the subroutine `Op_make`:

```
do i  = 1,Ndim
   call Op_make(Op_V(i,1),1)
enddo
```

For each lattice site i, the matrices $\boldsymbol{P}$ are of dimension $1 \times N_{\rm dim}$ and have only one non-vanishing entry. Thereby we can set:

```
Do i = 1,Ndim
   Op_V(i,1)%P(1)   = i
   Op_V(i,1)%O(1,1) = cmplx(1.d0,0.d0, kind(0.D0))
   Op_V(i,1)%g      = sqrt(cmplx(-dtau*ham_U/dble(N_SUN),0.D0,kind(0.D0)))
   Op_V(i,1)%alpha  = cmplx(-0.5d0,0.d0, kind(0.D0))
   Op_V(i,1)%type   = 2
   Call Op_set( Op_V(i,1) )
Enddo
```

so as to completely define the interaction term.

5.1.2 Observables

At this point, all the information for the simulation to start has been provided. The code will sequentially go through the operator list `Op_V` and update the fields. Between time slices `LOBS_ST` and `LOBS_EN` the main program will call the routine `Obser(GR,Phase,Ntau)` which is provided by the user and handles equal time correlation functions. If `Ltau=1` the main program will call the routine `ObserT(NT, GT0,G0T,G00,GTT, PHASE)` which is again provided by the user and handles imaginary-time displaced correlation functions.

The user will have to implement the observables they want to compute. Here we will describe how to proceed.

5.1.2.1 Allocating space for the observables: `Call Alloc_obs(Ltau)`

For four scalar or vector observables, the user will have to declare the following:

```
Allocate ( Obs_scal(4) )
Do I = 1,Size(Obs_scal,1)
   select case (I)
   case (1)
      N = 2;  Filename ="Kin"
   case (2)
      N = 1;  Filename ="Pot"
   case (3)
      N = 1;  Filename ="Part"
   case (4)
      N = 1,  Filename ="Ener"
   case default
      Write(6,*) ' Error in Alloc_obs '
   end select
   Call Obser_Vec_make(Obs_scal(I),N,Filename)
enddo
```

Here, `Obs_scal(1)` contains a vector of two observables so as to account for the x- and y-components of the kinetic energy for example.

For equal time correlation functions we allocate `Obs_eq` of type `Obser_Latt`. Here we include the calculation of spin-spin and density-density correlation functions alongside equal time Green functions.

```
Allocate ( Obs_eq(4) )
Do I = 1,Size(Obs_eq,1)
   select case (I)
   case (1)
      Ns = Latt%N; No = Norb;  Filename ="Green"
   case (2)
      Ns = Latt%N; No = Norb;  Filename ="SpinZ"
   case (3)
      Ns = Latt%N; No = Norb;  Filename ="SpinXY"
   case (4)
      Ns = Latt%N; No = Norb;  Filename ="Den"
   case default
      Write(6,*) ' Error in Alloc_obs '
   end select
   Nt = 1
   Call Obser_Latt_make(Obs_eq(I),Ns,Nt,No,Filename)
enddo
```

For the Hubbard model `Norb = 1` and for equal time correlation functions `Nt = 1`. If `Ltau = 1` then the code will allocate space for time displaced quantities. The same structure as for equal time correlation functions will be used albeit with `Nt = Ltrot + 1`. At the beginning of each bin, the main program will set the bin observables to zero by calling the routine `Init_obs(Ltau)`. The user does not have to edit this routine.

5.1.2.2 Measuring equal time observables: `Obser(GR,Phase,Ntau)`

The equal time Green function,

$$\texttt{GR(x,y,}\sigma\texttt{)} = \langle c_{x,\sigma} c^{\dagger}_{y,\sigma} \rangle, \tag{120}$$

the phase factor `phase` [Eq. (87)], and time slice `Ntau` are provided by the main program.

Here, x and y label both unit cell as well as the orbital within the unit cell. For the Hubbard model described here, x corresponds to the unit cell. The Green function does not depend on the color index, and is diagonal in flavor. For the $SU(2)$ symmetric implementation there is only one flavor, $\sigma = 1$ and the Green function is independent on the spin index. This renders the calculation of the observables particularly easy.

An explicit calculation of the potential energy $\langle U \sum_{\vec{i}} \hat{n}_{\vec{i},\uparrow} \hat{n}_{\vec{i},\downarrow} \rangle$ reads

```
Obs_scal(2)%N     = Obs_scal(2)%N + 1
Obs_scal(2)%Ave_sign = Obs_scal(2)%Ave_sign + Real(ZS,kind(0.d0))
Do i = 1,Ndim
   Obs_scal(2)%Obs_vec(1) = Obs_scal(2)%Obs_vec(1) + (1-GR(i,i,1))**2 * Ham_U * ZS * ZP
Enddo
```

Here `ZS` $=$ sign(C) [see Eq. (23)], `ZP` $= \frac{e^{-S(C)}}{\mathrm{Re}[e^{-S(C)}]}$ [see Eq. (87)] and `Ham_U` corresponds to the Hubbard-U term.

Equal time correlations are also computed in this routine. As an explicit example, we consider the equal time density-density correlation:

$$\langle n_{\vec{i},\alpha} n_{\vec{j},\beta} \rangle - \langle n_{\vec{i},\alpha} \rangle \langle n_{\vec{j},\beta} \rangle \ . \tag{121}$$

For the calculation of such quantities, it is convenient to define:

$$\texttt{GRC(x,y,s)} = \delta_{x,y} - \texttt{GR(y,x,s)} \tag{122}$$

such that `GRC(x,y,s)` corresponds to $\langle \langle \hat{c}^{\dagger}_{x,s} \hat{c}_{y,s} \rangle \rangle$. In the program code, the calculation of the equal time density-density correlation function looks as follows:

```
Obs_eq(4)%N     = Obs_eq(4)%N + 1       ! Even if it is redundant, each observable
                                        ! carries its own counter and sign.
Obs_eq(4)%Ave_sign = Obs_eq(4)%Ave_sign + Real(ZS,kind(0.d0))
Do I1 = 1,Ndim
   I    = List(I1,1)                    ! = I1  (The Hubbard model  on the square
   no_I = List(I1,2)                    ! = 1   lattice has one orbital per unit-cell)
   Do J1 = 1,Ndim
      J    = List(J1,1)
      no_J = List(J1,2)
      imj = latt%imj(I,J)
      Obs_eq(4)%Obs_Latt(imj,1,no_I,no_J) =  Obs_eq(4)%Obs_Latt(imj,1,no_I,no_J) + &
                     &     (    GRC(I1,I1,1) * GRC(J1,J1,1) * N_SUN * N_SUN      + &
                     &          GRC(I1,J1,1) * GR(I1,J1,1) * N_SUN   ) * ZP * ZS
   Enddo
   Obs_eq(4)%Obs_Latt0(no_I) = Obs_eq(4)%Obs_Latt0(no_I) + GRC(I1,I1,1) * N_SUN*ZP*ZS
Enddo
```

Note that we consider the square lattice of the single site Hubbard model as a special case of a multi-orbital problem as described in Sec. 5.3.1 At the end of each bin the main program will call the routine `Pr_obs(LTAU)`. This routine will append the result of the bins in the specified file, with appropriate suffix.

5.1.2.3 Measuring time displaced observables: `ObserT(NT, GT0,G0T,G00,GTT, PHASE)`

This subroutine is called by the main program at the beginning of each sweep, provided that `LTAU` is set to unity. `NT` runs from `0` to `Ltrot` and denotes the imaginary time difference. For a given time displacement, the main program provides:

$$\begin{aligned}
\texttt{GT0(x,y,s)} &= \langle\langle \hat{c}_{x,s}(Nt\Delta\tau)\hat{c}^\dagger_{y,s}(0)\rangle\rangle = \langle\langle \mathcal{T}\hat{c}_{x,s}(Nt\Delta\tau)\hat{c}^\dagger_{y,s}(0)\rangle\rangle \\
\texttt{G0T(x,y,s)} &= -\langle\langle \hat{c}^\dagger_{y,s}(Nt\Delta\tau)\hat{c}_{x,s}(0)\rangle\rangle = \langle\langle \mathcal{T}\hat{c}_{x,s}(0)\hat{c}^\dagger_{y,s}(Nt\Delta\tau)\rangle\rangle \\
\texttt{G00(x,y,s)} &= \langle\langle \hat{c}_{x,s}(0)\hat{c}^\dagger_{y,s}(0)\rangle\rangle \\
\texttt{GTT(x,y,s)} &= \langle\langle \hat{c}_{x,s}(Nt\Delta\tau)\hat{c}^\dagger_{y,s}(Nt\Delta\tau)\rangle\rangle
\end{aligned} \tag{123}$$

In the above we have omitted the color index since the Green functions are color independent. The time displaced spin-spin correlations $4\langle\langle \hat{S}^z_{\vec{i}}(\tau)\hat{S}^z_{\vec{j}}(0)\rangle\rangle$ are thereby given by:

$$4\langle\langle \hat{S}^z_{\vec{i}}(\tau)\hat{S}^z_{\vec{j}}(0)\rangle\rangle = -2\,\texttt{G0T(J1,I1,1)}\;\texttt{GT0(I1,J1,1)} \quad . \tag{124}$$

Note that the above holds for the $SU(2)$ HS transformation discussed in this chapter. The handling of time displaced correlation functions is identical to that of equal time correlations.

5.1.3 Numerical precision

The directory `Examples/Hubbard_SU2_Square` contains an example simulation of the 4×4 Hubbard model at $U/t = 4$ and $\beta t = 10$. Information on the numerical stability is included in the following lines of the corresponding file `info`:

```
Precision Green  Mean, Max :    1.2918865817224671E-014   4.0983018995027644E-011
Precision Phase, Max       :    5.0272908791449966E-012
Precision tau    Mean, Max :    8.4596701790588625E-015   3.5033530012121281E-011
```

showing the mean and maximum difference between the *wrapped* and from scratched computed equal and time displaced Green functions [6]. A stable code should produce results where the mean difference is smaller than the stochastic error. The above example shows a very stable simulation since the Green function is of order one.

5.2 The M_z-Hubbard model on a square lattice

The Hubbard Hamiltonian can equally be written as:

$$\mathcal{H} = \sum_{\sigma=1}^{2} \sum_{x,y=1}^{N_{unit\ cells}} c^{\dagger}_{x\sigma} T_{x,y} c_{y\sigma} - \frac{U}{2} \sum_{x} \left[c^{\dagger}_{x,\uparrow} c_{x\uparrow} - c^{\dagger}_{x,\downarrow} c_{x\downarrow} \right]^2 . \tag{125}$$

We can make contact with the general form of the Hamiltonian (see Eq. 2) by setting: $N_{\rm fl} = 2$, $N_{\rm col} \equiv$ `N_SUN` $= 1$, $M_T = 1$, $T^{(ks)}_{xy} = T_{x,y}$, $M_V = N_{\text{unit cell}}$, $U_k = \frac{U}{2}$, $V^{(k,s=1)}_{xy} = \delta_{x,y}\delta_{x,k}$, $V^{(k,s=2)}_{xy} = -\delta_{x,y}\delta_{x,k}$, $\alpha_{ks} = 0$ and $M_I = 0$. The coupling of the HS fields to the z-component of the magnetization breaks the $SU(2)$ spin symmetry. Nevertheless the z-component of the spin remains a good quantum number such that the imaginary-time propagator – for a given HS field – is block diagonal in this quantum number. This corresponds to the flavor index which runs from one to two labelling spin up and spin down degrees of freedom. In the parameter file listed in Sec. 3.3.1 setting the model variable to `Hubbard_Mz` will carry out the simulation in the above representation. With respect to the $SU(2)$ case, the changes required in the `Hamiltonian_Examples.F90` module are minimal and essentially effect only the interaction term, and the calculation of observables. We note that in this formulation the hopping matrix can be flavor dependent such that a Zeeman magnetic field can be introduced. If the chemical potential is set to zero, this will not generate a negative sign problem [33, 76, 77]. A sample run for this model can be found in `Examples/Hubbard_Mz_Square/`.

5.2.1 The interaction term: `Call Ham_V`

The interaction term is now given by:

```
Allocate(Op_V(Ndim,N_FL))
do nf = 1,N_FL
   do i  = 1, Ndim
      Call Op_make(Op_V(i,nf),1)
   enddo
enddo
Do nf = 1,N_FL
   nc = 0
   X = 1.d0
   if (nf == 2) X = -1.d0
   Do i = 1,Ndim
      nc = nc + 1
      Op_V(nc,nf)%P(1) = I
      Op_V(nc,nf)%O(1,1) = cmplx(1.d0, 0.d0, kind(0.D0))
      Op_V(nc,nf)%g      = X*SQRT(CMPLX(DTAU*ham_U/2.d0, 0.D0, kind(0.D0)))
      Op_V(nc,nf)%alpha  = cmplx(0.d0, 0.d0, kind(0.D0))
      Op_V(nc,nf)%type   = 2
      Call Op_set( Op_V(nc,nf) )
   Enddo
Enddo
```

In the above, one will see explicitly that there is a sign difference between the coupling of the HS field in the two flavor sectors.

5.2.2 The measurements: `Call Obser, Call ObserT`

Since the spin up and spin down Green functions differ for a given HS configuration, the Wick decomposition will take a different form. In particular, the equal time spin-spin correlation functions $4\langle\langle \hat{S}^z_i \hat{S}^z_j \rangle\rangle$ calculated in the subroutine `Obser` will take the form:

$$4\langle\langle \hat{S}^z_x \hat{S}^z_y \rangle\rangle = \quad \texttt{GRC(x,y,1) * GR(x,y,1) + GRC(x,y,2) * GR(x,y,2) +}$$
$$\texttt{(GRC(x,x,2) - GRC(x,x,1))*(GRC(y,y,2) - GRC(y,y,1))}$$

Here, `GRC` is defined in Eq. 122. Equivalent changes will have to be carried out for other equal time and time displaced observables.

Apart from these modifications, the program will run in exactly the same manner as for the $SU(2)$ case.

5.2.3 Numerical precision

The directory `Examples/Hubbard_Mz_Square` contains an example simulation of the 4×4 Hubbard model at $U/t = 4$ and $\beta t = 10$. Information on the numerical stability is included in the following lines of the corresponding file `info`:

```
Precision Green  Mean, Max :    5.0823874429126405E-011   5.8621144596315844E-006
Precision Phase, Max       :    0.0000000000000000
Precision tau    Mean, Max :    1.5929357848647394E-011   1.0985132530727526E-005
```

This is still an excellent precision but nevertheless choosing a HS field which couples to the z-component of the magnetization apparently leads to numerical results that are a couple of order of magnitudes less precise than a HS decomposition coupling to the charge (compare with Sec. 5.1.3).

5.3 The $SU(2)$-Hubbard model on the honeycomb lattice

The Hamilton module `Hamiltonian_Examples.F90` can also carry out simulations for the Hubbard model on the Honeycomb lattice by setting in the parameter file `Lattice_type="Honeycomb"` (see Sec. 3.3.1). A sample run for this model can be found in `Examples/Hubbard_SU2_Honeycomb/`.

5.3.1 Working with multi-orbital unit cells: `Call Ham_Latt`

This model is an example of a multi-orbital unit cell, and the aim of this section is to document how to implement this in the code. The Honeycomb lattice is a triangular Bravais lattice with two orbitals per unit cell. The routine `Ham_Latt` will set:

```
Norb    = 2
N_coord = 3
a1_p(1) =  1.D0   ; a1_p(2) =  0.d0
a2_p(1) =  0.5D0  ; a2_p(2) =  sqrt(3.D0)/2.D0
L1_p    =  dble(L1) * a1_p
L2_p    =  dble(L2) * a2_p
```

and then call `Make_Lattice(L1_p,L2_p,a1_p,a2_p,Latt)` so as to generate the triangular lattice. The coordination number of this lattice is `N_coord=3` and the number of orbitals per unit cell corresponds to `NORB=2`. The total number of orbitals is thereby: $N_{\rm dim}$`=Latt%N*NORB`. To easily keep track of the orbital and unit cell, we define a super-index as shown below:

```
Allocate (List(Ndim,2), Invlist(Latt%N,Norb))
nc = 0
Do I = 1,Latt%N                                  ! Unit-cell index
   Do no = 1,Norb                                ! Orbital index
      nc = nc + 1                                ! Super-index labeling unit cell
                                                 ! and orbital
      List(nc,1) = I                             ! Unit-cell  of super index  nc
      List(nc,2) = no                            ! Orbital of super index nc
      Invlist(I,no) = nc                         ! Super-index for given  unit cell
                                                 ! and orbital
  Enddo
Enddo
```

With the above lists one can run through all the orbitals and at each time keep track of the unit-cell and orbital index. We note that when translation symmetry is completely absent one can work with a single unit cell, and the number of orbitals will then correspond to the number of lattice sites.

5.3.2 The hopping term: `Call Ham_Hop`

Some care has to be taken when setting the hopping matrix. In the Hamilton module `Hamiltonian_Examples.F90` we do this in the following way:

```
DO I = 1, Latt%N                                  ! Loop over unit cell
   do no = 1,Norb                                 ! Runs over orbitals and
                                                  ! sets the chemical potential
      I1 = invlist(I,no)
      Op_T(nc,n)%O(I1 ,I1) = cmplx(-Ham_chem, 0.d0, kind(0.D0))
   enddo
   I1 = Invlist(I,1)                              ! Orbital A of unit cell I
   Do nc1 = 1,N_coord                             ! Loop over coordination  number
      select case (nc1)
      case (1)
         J1 = invlist(I,2)                        ! Orbital B of unit cell i
      case (2)
         J1 = invlist(Latt%nnlist(I,1,-1),2)      ! Orbital B of unit cell i + a_1 - a_2
      case (3)
         J1 = invlist(Latt%nnlist(I,0,-1),2)      ! Orbital B of unit cell i - a_2
      case default
         Write(6,*) ' Error in  Ham_Hop '
      end select
      Op_T(nc,n)%O(I1,J1) = cmplx(-Ham_T,    0.d0, kind(0.D0))
      Op_T(nc,n)%O(J1,I1) = cmplx(-Ham_T,    0.d0, kind(0.D0))
   Enddo
Enddo
```

As apparent from the above, hopping matrix elements are non-zero only between the A and B sublattices.

5.3.3 Observables: `Call Obser`, `Call ObserT`

In the multi-orbital case, the correlation functions have additional orbital indices. This is automatically taken care of in the routines `Call Obser` and `Call ObserT` since we have already considered the Hubbard model on the square lattice to correspond to a multi-orbital unit cell albeit with the special choice of one orbital per unit cell.

5.4 The $SU(2)$-Hubbard model on a square lattice coupled to a transverse Ising field

The model we consider here is very similar to the above, but has an additional coupling to a transverse field:

$$\begin{aligned}\mathcal{H} = & \sum_{\sigma=1}^{2}\sum_{x,y} c^\dagger_{x\sigma} T_{x,y} c_{y\sigma} + \frac{U}{2}\sum_x \left[\sum_{\sigma=1}^{2}\left(c^\dagger_{x\sigma}c_{x\sigma} - 1/2\right)\right]^2 + \xi \sum_{\sigma,\langle x,y\rangle} \hat{Z}_{\langle x,y\rangle}\left(c^\dagger_{x\sigma}c_{y\sigma} + h.c.\right) \\ & -h\sum_{\langle x,y\rangle}\hat{X}_{\langle x,y\rangle} - J \sum_{\langle\langle x,y\rangle\langle x',y'\rangle\rangle} \hat{Z}_{\langle x,y\rangle}\hat{Z}_{\langle x',y'\rangle} \end{aligned} \tag{126}$$

We can make contact with the general form of the Hamiltonian by setting: $N_{\mathrm{fl}} = 1$, $N_{\mathrm{col}} \equiv$ `N_SUN` $= 2$, $M_T = 1$, $T^{(ks)}_{xy} = T_{x,y}$, $M_V = N_{\text{unit cell}} \equiv N_{\mathrm{dim}}$, $U_k = -\frac{U}{2}$, $V^{(ks)}_{xy} = \delta_{x,y}\delta_{x,k}$, $\alpha_{ks} = -\frac{1}{2}$ and $M_I = 2N_{\text{unit cell}}$. The last two terms of the above Hamiltonian describe a transverse Ising field model on the bonds of the square lattice. This type of Hamiltonian has recently been extensively discussed [20, 78, 17]. Here we adopt the notation of Ref. [17]. Note that $\langle\langle x,y\rangle\langle x',y'\rangle\rangle$ denotes nearest neighbor bonds. The modifications required to generalize the Hubbard model code to the above model are two-fold.

First, one has to specify the function `Real(Kind=8)functionS0(n,nt)`, and second, modify the interaction `Call Ham_V`.

A sample run for this model can be found in `Examples/Hubbard_SU2_Ising_Square/`.

5.4.1 The Ising term.

Since the Ising field lives on bonds we have to provide a data structure defining this quantity. A bond has an anchor site as well as an orientation. The routine `Setup_Ising_action` initializes the arrays `L_bond` and `L_bond_inv` that contain this information.

```
nc = 0
Do n_orientation = 1,N_coord
Do I = 1, Latt%N
   nc = nc + 1
   L_bond(I,n_orientation) = nc
   L_bond_inv(nc,1) = I
   L_bond_inv(nc,2) = n_orientation
enddo
enddo
```

The two legs of the bond are given by the anchor $\vec{I}$ and $\vec{I} + \vec{a}_{n_{orientation}}$

5.4.2 The interaction term: `Call Ham_V`

The dimension of `Op_V` is now $(M_I + M_V) \times N_{\rm fl} = ((N_{coord} + 1)N_{\rm dim}) \times 1$ since each site has $N_{coord} = 2$ bonds for the square lattice.

```
do i  = 1,N_coord*Ndim                    ! Runs over bonds for Ising interaction.
  call Op_make(Op_V(i,1),2)
enddo
do i  =  N_coord*Ndim+1, (N_coord+1)*Ndim  ! Runs over sites for Hubbard interaction.
  call Op_make(Op_V(i,1),1)
enddo
```

The first `N_coord*Ndim` operators run through the $2N$ bonds of the square lattice and are given by:

```
Do nc = 1,Ndim*N_coord                    ! Runs over bonds. Coordination number = 2.
                                          ! For the square lattice Ndim = Latt%N

   I1 = L_bond_inv(nc,1)                  ! Anchor of the bond
                                          ! L_bond_inv is setup in Setup_Ising_action
   if ( L_bond_inv(nc,2)  == 1 ) I2 = Latt%nnlist(I1,1,0)  ! Second site of the bond
   if ( L_bond_inv(nc,2)  == 2 ) I2 = Latt%nnlist(I1,0,1)
   Op_V(nc,1)%P(1) = I1
   Op_V(nc,1)%P(2) = I2
   Op_V(nc,1)%O(1,2) = cmplx(1.d0 ,0.d0, kind(0.D0))
   Op_V(nc,1)%O(2,1) = cmplx(1.d0 ,0.d0, kind(0.D0))
   Op_V(nc,1)%g      = cmplx(-dtau*Ham_xi,0.D0,kind(0.D0))
   Op_V(nc,1)%alpha  = cmplx(0d0,0.d0, kind(0.D0))
   Op_V(nc,1)%type   = 1
Enddo
```

Here, `ham_xi` defines the coupling strength between the Ising and fermion degree of freedom. As for the Hubbard case, the last `Ndim` operators read:

```
nc = N_coord*Ndim
Do i = 1, Ndim
    nc = nc + 1
    Op_V(nc,1)%P(1)   = i
    Op_V(nc,1)%O(1,1) = cmplx(1.d0  ,0.d0, kind(0.D0))
    Op_V(nc,1)%g      = sqrt(cmplx(-dtau*ham_U/(DBLE(N_SUN)), 0.D0, kind(0.D0)))
    Op_V(nc,1)%alpha  = cmplx(-0.5d0,0.d0, kind(0.D0))
```

```
    Op_V(nc,1)%type   = 2
Enddo
```

5.4.3 The function `Real (Kind=8) function S0(n,nt)`

As mentioned above, a configuration now includes both HS spins and Ising spins and is given by

$$C = \{s_{i,\tau}, l_{j,\tau} \text{ with } i = 1 \cdots M_I, j = 1 \cdots M_V, \tau = 1, L_{Trotter}\} \ . \tag{127}$$

This configuration is stored in the integer array `nsigma(M_V + M_I, Ltrot)`. With the above ordering of Hubbard and Ising interaction terms, and a for a given imaginary time, the first `2*Ndim` fields correspond to the Ising interaction and the next `Ndim` ones to the Hubbard interaction. The first argument of the function `S0`, namely `n`, corresponds to the index of the operator string `Op_V(n,1)`. If `Op_V(n,1)%type = 2` then `S0(n,nt)` returns 1. Note that `type=2` refers to spins that stem from a HS transformation. If however `Op_V(n,1)%type = 1` then function `S0` returns

$$\frac{e^{-S_{0,I}\left(s_{1,\tau},\cdots,-s_{n,\tau},\cdots s_{M_I,\tau}\right)}}{e^{-S_{0,I}\left(s_{1,\tau},\cdots,s_{n,\tau},\cdots s_{M_I,\tau}\right)}} \tag{128}$$

That is, if $n \leq 2 * \texttt{Ndim}$, `S0(n,nt)` returns the ratio of the new weight to the old weight of the Ising Hamiltonian upon flipping a single Ising spin $s_{n,\tau}$. Otherwise, `S0(n,nt)` returns unity.

6 Miscellaneous

6.1 Other models

The aim of this section is to briefly mention a small selection of other models that can be studied using the QMC code of the ALF project.

6.1.1 Kondo lattice model

Simulating the Kondo lattice with the QMC code of the ALF project requires rewriting of the model along the lines of Refs. [18, 19, 79]. Adopting the notation of these articles, the Hamiltonian that one will simulate reads:

$$\hat{\mathcal{H}} = \underbrace{-t \sum_{\langle \vec{i},\vec{j}\rangle,\sigma} \left(\hat{c}^{\dagger}_{\vec{i},\sigma}\hat{c}_{\vec{j},\sigma} + \text{H.c.}\right)}_{\equiv\hat{\mathcal{H}}_t} - \frac{J}{4}\sum_{\vec{i}}\left(\sum_{\sigma}\hat{c}^{\dagger}_{\vec{i},\sigma}\hat{f}_{\vec{i},\sigma} + \hat{f}^{\dagger}_{\vec{i},\sigma}\hat{c}_{\vec{i},\sigma}\right)^2 + \underbrace{\frac{U}{2}\sum_{\vec{i}}\left(\hat{n}^{f}_{\vec{i}} - 1\right)^2}_{\equiv\hat{\mathcal{H}}_U}. \tag{129}$$

This form is included in the general Hamiltonian (2) such that the above Hamiltonian can be implemented in our program package. The relation to the Kondo lattice model follows from expanding the square of the hybridization to obtain:

$$\hat{\mathcal{H}} = \hat{\mathcal{H}}_t + J\sum_{\vec{i}}\left(\hat{\vec{S}}^{c}_{\vec{i}}\cdot\hat{\vec{S}}^{f}_{\vec{i}} + \hat{\eta}^{z,c}_{\vec{i}}\cdot\hat{\eta}^{z,f}_{\vec{i}} - \hat{\eta}^{x,c}_{\vec{i}}\cdot\hat{\eta}^{x,f}_{\vec{i}} - \hat{\eta}^{y,c}_{\vec{i}}\cdot\hat{\eta}^{y,f}_{\vec{i}}\right) + \hat{\mathcal{H}}_U. \tag{130}$$

where the η-operators relate to the spin-operators via a particle-hole transformation in one spin sector:

$$\hat{\eta}^{\alpha}_{\vec{i}} = \hat{P}^{-1}\hat{S}^{\alpha}_{\vec{i}}\hat{P} \ \text{ with } \ \hat{P}^{-1}\hat{c}_{\vec{i},\uparrow}\hat{P} = (-1)^{i_x+i_y}\hat{c}^{\dagger}_{\vec{i},\uparrow} \ \text{ and } \ \hat{P}^{-1}\hat{c}_{\vec{i},\downarrow}\hat{P} = \hat{c}_{\vec{i},\downarrow} \tag{131}$$

Since the $\hat{\eta}^{f}$- and $\hat{S}^{f}$-operators do not alter the parity $[(-1)^{\hat{n}^{f}_{\vec{i}}}]$ of the f-sites,

$$\left[\hat{\mathcal{H}}, \hat{\mathcal{H}}_U\right] = 0. \tag{132}$$

Thereby, and for positive values of U , doubly occupied or empty f-sites – corresponding to even parity sites – are suppressed by a Boltzmann factor $e^{-\beta U/2}$ in comparison to odd parity sites. Choosing βU adequately essentially allows to restrict the Hilbert space to odd parity f-sites. In this Hilbert space $\hat{\eta}^{x,f} = \hat{\eta}^{y,f} = \hat{\eta}^{z,f} = 0$ such that the Hamiltonian (129) reduces to the Kondo lattice model.

An implementation of the Kondo Lattice model on the Honeycomb lattice with additional z-z frustration considered in Ref. [80],

$$\hat{H}_{\text{Spin}} = J^z \sum_{\langle\langle i,j \rangle\rangle} \hat{S}_i^z \hat{S}_j^z, \quad \hat{H}_{\text{Fermion}} = -t \sum_{\langle i,j \rangle,\sigma} \hat{c}_{i\sigma}^\dagger \hat{c}_{j\sigma}$$
$$\hat{H}_{\text{Kondo}} = J^{\text{K}} \sum_i \frac{1}{2} \hat{\boldsymbol{c}}_i^\dagger \boldsymbol{\sigma} \hat{\boldsymbol{c}}_i \cdot \hat{\boldsymbol{S}}_i \,, \tag{133}$$

can be found in the `Hamiltonian_Kondo_Honey.F90`

6.1.2 $SU(N)$ Hubbard-Heisenberg models

$SU(2N)$ Hubbard-Heisenberg [23, 24] models can be written as:

$$\hat{\mathcal{H}} = \underbrace{-t \sum_{\langle \vec{i},\vec{j} \rangle} \left(\vec{\hat{c}}_{\vec{i}}^\dagger \vec{\hat{c}}_{\vec{j}} + \text{H.c.} \right)}_{\equiv \hat{\mathcal{H}}_t} \underbrace{-\frac{J}{2N} \sum_{\langle \vec{i},\vec{j} \rangle} \left(\hat{D}_{\vec{i},\vec{j}}^\dagger \hat{D}_{\vec{i},\vec{j}} + \hat{D}_{\vec{i},\vec{j}} \hat{D}_{\vec{i},\vec{j}}^\dagger \right)}_{\equiv \hat{\mathcal{H}}_J} + \underbrace{\frac{U}{N} \sum_{\vec{i}} \left(\vec{\hat{c}}_{\vec{i}}^\dagger \vec{\hat{c}}_{\vec{i}} - \frac{N}{2} \right)^2}_{\equiv \hat{\mathcal{H}}_U} \tag{134}$$

Here, $\vec{\hat{c}}_{\vec{i}}^\dagger = (\hat{c}_{\vec{i},1}^\dagger, \hat{c}_{\vec{i},2}^\dagger, \cdots, \hat{c}_{\vec{i},N}^\dagger)$ is an N-flavored spinor, and $\hat{D}_{\vec{i},\vec{j}} = \vec{\hat{c}}_{\vec{i}}^\dagger \vec{\hat{c}}_{\vec{j}}$. To use the QMC code of the ALF project to simulate this model, one will rewrite the J-term as a sum of perfect squares,

$$\hat{\mathcal{H}}_J = -\frac{J}{4N} \sum_{\langle \vec{i},\vec{j} \rangle} \left(\hat{D}_{\langle \vec{i},\vec{j} \rangle}^\dagger + \hat{D}_{\langle \vec{i},\vec{j} \rangle} \right)^2 - \left(\hat{D}_{\langle \vec{i},\vec{j} \rangle}^\dagger - \hat{D}_{\langle \vec{i},\vec{j} \rangle} \right)^2, \tag{135}$$

so to manifestly bring it into the form of the general Hamiltonian(2). It is amusing to note that setting the hopping $t = 0$, charge fluctuations will be suppressed by the Boltzmann factor $e^{-\beta U/N \left(\vec{\hat{c}}_{\vec{i}}^\dagger \vec{\hat{c}}_{\vec{i}} - \frac{N}{2} \right)^2}$ since in this case $\left[\hat{\mathcal{H}}_J, \hat{\mathcal{H}}_U \right] = 0$. This provides a route to use the auxiliary field QMC algorithm to simulate – free of the sign problem – $SU(2N)$ Heisenberg models in the self-adjoint antisymmetric representation [6] For odd values of N recent progress in our understanding of the origins of the sign problem [36] allows us to simulate a set of non-trivial Hamiltonians [81, 17], without encountering the sign problem.

6.1.3 Hubbard model in the canonical ensemble

To simulate the Hubbard model in the canonical ensemble one can add the constraint:

$$\hat{\mathcal{H}} = \hat{\mathcal{H}_{tU}} + \underbrace{\lambda \left(\hat{N} - N \right)^2}_{\equiv \hat{H}_\lambda} \tag{136}$$

In the limit $\lambda \to \infty$, the uniform charge fluctuations,

$$S(\vec{q} = 0) = \sum_{\vec{r}} \left[\langle \hat{n}_{\vec{r}} \hat{n}_{\vec{0}} \rangle - \langle \hat{n}_{\vec{r}} \rangle \langle \hat{n}_{\vec{0}} \rangle \right] \tag{137}$$

are suppressed and the grand-canonical simulation maps onto a canonical one. To implement this in the QMC code of the ALF project, we have adopted the following strategy. Since $\left(\hat{N} - N \right)^2$ effectively corresponds to a long-range interaction one may face the issue that the acceptance rate of a single HS flip becomes very small on large lattices. To circumvent this problem we have used the following decomposition:

$$e^{-\beta \hat{H}} = \prod_{\tau=1}^{L_{\text{Trotter}}} \left[e^{-\Delta\tau \hat{H}_t} e^{-\Delta\tau \hat{H}_U} \underbrace{e^{-\frac{\Delta\tau}{n_\lambda} \hat{H}_\lambda} \cdots e^{-\frac{\Delta\tau}{n_\lambda} \hat{H}_\lambda}}_{n_\lambda\text{-times}} \right]. \tag{138}$$

Thereby, we need n_λ fields per time slice to impose the constraint. Since for each field the coupling constant is suppressed by a factor n_λ, we can monitor the acceptance. An implementation of this program can be found in `Prog/Hamiltonian_Hub_Canonical.F90` and a test run in the directory `Examples/Hubbard_Mz_Square_Can`

[6] This corresponds to a Young tableau with single column and $N/2$ rows.

6.1.4 Z_2 slave spin formualtion of the Hubbard model

In this subsection, we demonstrate that the code can be used to simulate the attractive Hubbard model in the Z_2-slave spin formulation [82].

$$\hat{H} = -t \sum_{\langle \vec{i},\vec{j}\rangle,\sigma} \hat{c}^{\dagger}_{\vec{i},\sigma} \hat{c}_{\vec{j},\sigma} - U \sum_{i} \left(\hat{n}_{\vec{i},\uparrow} - 1/2\right)\left(\hat{n}_{\vec{i},\downarrow} - 1/2\right) \tag{139}$$

In the Z_2 slave spin representation, the physical fermion, $\hat{c}_{\vec{i},\sigma}$, is fractionalized into an Ising spin carrying Z_2 charge and a fermion, $\hat{f}_{\vec{i},\sigma}$, carrying Z_2 and global $U(1)$ charge:

$$\hat{c}^{\dagger}_{\vec{i},\sigma} = \hat{\tau}^{z}_{\vec{i}} \hat{f}^{\dagger}_{\vec{i},\sigma}. \tag{140}$$

To ensure that we remain in the correct Hilbert space, the constraint:

$$\hat{\tau}^{x}_{\vec{i}} - (-1)^{\sum_{\sigma} \hat{f}^{\dagger}_{i,\sigma} \hat{f}_{i,\sigma}} = 0 \tag{141}$$

has to be impose locally. Since $\left(\tau^{x}_{\vec{i}}\right)^2 = 1$ it is equivalent to

$$\hat{Q}_{\vec{i}} = \tau^{x}_{\vec{i}} (-1)^{\sum_{\sigma} \hat{f}^{\dagger}_{\vec{i},\sigma} \hat{f}_{\vec{i},\sigma}} = 1. \tag{142}$$

In the Z_2 slave spin representation the Hubbard model now reads:

$$\hat{H}_{Z_2} = -t \sum_{\langle \vec{i},\vec{j}\rangle,\sigma} \hat{\tau}^{z}_{\vec{i}} \hat{\tau}^{z}_{\vec{j}} \hat{f}^{\dagger}_{\vec{i},\sigma} \hat{f}_{\vec{j},\sigma} - \frac{U}{4} \sum_{\vec{i}} \hat{\tau}^{x}_{\vec{i}} \tag{143}$$

and one will readily see that the constraint will commute with Hamiltonian:

$$\left[\hat{H}_{Z_2}, \hat{Q}_{\vec{i}}\right] = 0. \tag{144}$$

One can foresee that the constraint will be dynamically imposed and that at $T = 0$ on a finite lattice both models should give the same results.

To implement this Hamiltonian in the ALF, it is convenient to carry out the variable substitution

$$\hat{Z}_{\langle \vec{i},\vec{j}\rangle} = \hat{\tau}^{z}_{\vec{i}} \hat{\tau}^{z}_{\vec{j}} \tag{145}$$

such that

$$\hat{\tau}^{x}_{\vec{i}} = \hat{X}_{\vec{i},\vec{i}+\vec{a}_x} \hat{X}_{\vec{i},\vec{i}-\vec{a}_x} \hat{X}_{\vec{i},\vec{i}+\vec{a}_y} \hat{X}_{\vec{i},\vec{i}-\vec{a}_y}. \tag{146}$$

Since there are twice as many bond variables as site variables, a constraint has to be imposed on the $\hat{Z}_{\langle \vec{i},\vec{j}\rangle}$ variables. In fact, it is easy to see that the flux per plaquette has to take a unit value:

$$\hat{Z}_{\vec{i},\vec{i}+\vec{a}_x} \hat{Z}_{\vec{i}+\vec{a}_x,\vec{i}+\vec{a}_x+\vec{a}_y} \hat{Z}_{\vec{i}+\vec{a}_x+\vec{a}_y,\vec{i}+\vec{a}_y} \hat{Z}_{\vec{i}+\vec{a}_y,\vec{i}} = 1 \quad \forall \quad \vec{i}. \tag{147}$$

Within this formulation the model takes the form:

$$\hat{H}_{Z_2} = -t \sum_{\langle \vec{i},\vec{j}\rangle,\sigma} \hat{Z}_{\langle \vec{i},\vec{j}\rangle} \hat{f}^{\dagger}_{\vec{i},\sigma} \hat{f}_{\vec{j},\sigma} - \frac{U}{4} \sum_{\vec{i}} \hat{X}_{\vec{i},\vec{i}+\vec{a}_x} \hat{X}_{\vec{i},\vec{i}-\vec{a}_x} \hat{X}_{\vec{i},\vec{i}+\vec{a}_y} \hat{X}_{\vec{i},\vec{i}-\vec{a}_y}. \tag{148}$$

The fermion part has formally the same from as in the Hubbard model coupled to a dynamical Ising field discussed in Sec. 5.4. There are however two important differences.

- The moves have to respect the constraint of Eq. 147. Thereby single spin flip terms are prohibited and the minimal move one can carry out on a given time slice is the following. We randomly choose a site $\vec{i}$ and propose a move where: $Z_{\vec{i},\vec{i}+\vec{a}_x} \to -Z_{\vec{i},\vec{i}+\vec{a}_x}$, $Z_{\vec{i},\vec{i}-\vec{a}_x} \to -Z_{\vec{i},\vec{i}-\vec{a}_x}$, $Z_{\vec{i},\vec{i}+\vec{a}_y} \to -Z_{\vec{i},\vec{i}+\vec{a}_y}$ and $Z_{\vec{i},\vec{i}-\vec{a}_y} \to -Z_{\vec{i},\vec{i}-\vec{a}_y}$. One can carry out such moves by using the global move in real space option presented in Secs. 2.2.3 and 3.3.1.

- The map from $\left\{\tau^z_{\vec{i}}\right\}$ to $\left\{Z_{\langle\vec{i},\vec{j}\rangle}\right\}$ is unique. The reverse however is valid only up to to a global sign. To pin down this sign (and thereby the relative signs between different time slices) we store per time slice the $Z_{\langle\vec{i},\vec{j}\rangle}$ fields as well as the value of the Ising field at a reference site $\tau^z_{\vec{i}=1}$. Within the ALF, this can be done by adding a dummy operator in the `Op_V` list which will carry this degree of freedom. With this extra degree of freedom we can switch between the two representations, without loosing any information. To compute the Ising part of the action it is certainly more transparent to work with the $\left\{\tau^z_{\vec{i}}\right\}$ variables. For the fermion determinant, the $\left\{Z_{\langle\vec{i},\vec{j}\rangle}\right\}$ are more convenient.

We have carried out some test simulations at half-filling and at low temperatures. The simulation can be found in directory `Examples/Z2_Slave` and the Hamiltonian in `Hamiltonian_Z2_slave_spins.F90` . The simulations are carried out at half-filling such that particle-hole symmetry leads to

$$\langle \hat{Q}_{\vec{i}} \rangle_{H_{Z_2}} = 0. \tag{149}$$

However the simulations suggest that

$$\frac{1}{N}\sum_{i,j} \langle \hat{Q}_{\vec{i}} \hat{Q}_{\vec{j}} \rangle_{H_{Z_2}} = N \tag{150}$$

at low temperatures where N corresponds to size of the lattice. Note that this measurement is very noisy, and suffers from very long autocorrelation times. Thereby, the constraint is dynamically imposed and simulations of the attractive Hubbard model with `Hamiltonian_Examples.F90` should yield identical results. To test this, we have computed equal time Green functions:

$$\langle \hat{\tau}^z_{\vec{i}} \hat{f}^\dagger_{\vec{i},\sigma} \hat{\tau}^z_{\vec{j}} \hat{f}_{\vec{j},\sigma} \rangle_{H_{Z_2}} = \langle \hat{c}^\dagger_{\vec{i},\sigma} \hat{c}_{\vec{j},\sigma} \rangle_H \tag{151}$$

as obtained from the `Hamiltonian_Examples.F90` and `Hamiltonian_Z2_slave_spins.F90` codes. A test run for the 8×8 lattice at $U/t = 4$ and $\beta t = 40$ gives:

k	$\langle n_k \rangle_H$	$\langle n_k \rangle_{H_{Z_2}}$
(0,0)	1.93348548 ± 0.00011322	1.93336807 ± 0.00080473
(π/4,π/4)	1.90120688 ± 0.00014854	1.90107164 ± 0.00097029
(π/2,π/2)	0.99942957 ± 0.00091377	1.00000000 ± 0.00000000
(3π/4,3π/4)	0.09905425 ± 0.00015940	0.09892836 ± 0.00097029
(π,π)	0.06651452 ± 0.00011321	0.06663193 ± 0.00080473

Here a Trotter time step of $\Delta\tau t = 0.05$ was used so as to minimize the systematic error which should be different between the two codes.

This code was used in Ref. [83].

6.1.5 Z_2 gauge theory coupled to Z_2 matter.

The Hamiltonian we will consider here reads

$$\begin{aligned}\hat{H} &= -t_{Z_2}\sum_{\langle\vec{i},\vec{j}\rangle,\sigma}\hat{Z}_{\langle\vec{i},\vec{j}\rangle}\left(\hat{\Psi}^{\dagger}_{\vec{i},\sigma}\hat{\Psi}_{\vec{j},\sigma}+h.c.\right)-\mu\sum_{\vec{i},\sigma}\hat{\Psi}^{\dagger}_{\vec{i},\sigma}\hat{\Psi}_{\vec{i},\sigma}-g\sum_{\langle\vec{i},\vec{j}\rangle}\hat{X}_{\langle\vec{i},\vec{j}\rangle}+K\sum_{\square}\prod_{\langle\vec{i},\vec{j}\rangle\in\partial\square}\hat{Z}_{\langle\vec{i},\vec{j}\rangle}\\ &+J\sum_{\langle\vec{i},\vec{j}\rangle}\hat{\tau}^{z}_{\vec{i}}\hat{Z}_{\langle\vec{i},\vec{j}\rangle}\hat{\tau}^{z}_{\vec{j}}-h\sum_{\vec{i}}\hat{\tau}^{x}_{\vec{i}}-t\sum_{\langle\vec{i},\vec{j}\rangle,\sigma}\hat{\tau}^{z}_{\vec{i}}\hat{\tau}^{z}_{\vec{j}}\left(\hat{\Psi}^{\dagger}_{\vec{i},\sigma}\hat{\Psi}_{\vec{j},\sigma}+h.c.\right)\end{aligned} \tag{152}$$

Here the $\hat{\Psi}^{\dagger}_{\vec{i},\sigma}$ creates an orthogonal fermion with Z_2 and and electric charges. The implementation of this Hamiltonian can be found in the file `Hamiltonian_Z2_Matter.F90`. For this Hamiltonian, the the Z_2 local conservation law reads:

$$\hat{Q}_{\vec{i}}=(-1)^{\sum_{\sigma}\hat{\Psi}^{\dagger}_{\vec{i},\sigma}\hat{\Psi}_{\vec{j},\sigma}}\,\hat{\tau}^{x}_{\vec{i}}\,\hat{X}_{\vec{i},\vec{i}+\vec{a}_x}\hat{X}_{\vec{i},\vec{i}-\vec{a}_x}\hat{X}_{\vec{i},\vec{i}+\vec{a}_y}\hat{X}_{\vec{i}}. \tag{153}$$

The Hamiltonian was investigated in Ref. [84]. Here is a todo list.

- Include an attractive U-term. This breaks the O(4) symmetry down to SU(2)×SU(2) and selects the $\hat{Q}_{\vec{i}}=1$ sector. Note that a repulsive U should can also be included and should produce equivalent results under particle-hole symmetry.
- Include dynamics, so as to study the dynamics of the OSM to FL* phase. Note that the FL* phase will ultimately be unstable to the AFM* phase, but the energy scale is expected to be extremely small at small U.
- Include a projective version, with different left and right wave functions. The right imposes translation invariance and the left the constraint. That is, we choose the right trial wave function to be the ground state of

$$\hat{H}^{R}_{T}=-t\sum_{\langle\vec{i},\vec{j}\rangle,\sigma}\left(\hat{\Psi}^{\dagger}_{\vec{i},\sigma}\hat{\Psi}_{\vec{j},\sigma}+h.c.\right)-h\sum_{\vec{i}}\left(\hat{X}_{\vec{i},\vec{i}+\vec{a}_x}+\hat{X}_{\vec{i},\vec{i}+\vec{a}_y}+\tau^{x}_{\vec{i}}\right) \tag{154}$$

  and the left one to be the ground state of

$$\hat{H}^{L}_{T}=-U\sum_{\vec{i},\sigma}e^{i\vec{Q}\cdot\vec{i}}\hat{\Psi}^{\dagger}_{\vec{i},\sigma}\hat{\Psi}_{\vec{i},\sigma}-h\sum_{\vec{i}}\left(\hat{X}_{\vec{i},\vec{i}+\vec{a}_x}+\hat{X}_{\vec{i},\vec{i}+\vec{a}_y}+\tau^{x}_{\vec{i}}\right) \tag{155}$$

  with $\vec{Q}=(\pi,\pi)$.

6.1.6 Generalized t-V model

The example code `Hamiltonian_Examples.F90` contains an implementation of the SU(N) t-V model, given by:

$$\hat{H}_{tV}=-t\sum_{\langle\vec{i},\vec{j}\rangle}\sum_{\sigma=1}^{N}\left(\hat{c}^{\dagger}_{\vec{i},\sigma}\hat{c}_{\vec{j},\sigma}+h.c.\right)-\frac{V}{N}\sum_{\langle\vec{i},\vec{j}\rangle}\left(\sum_{\sigma=1}^{N}\left(\hat{c}^{\dagger}_{\vec{i},\sigma}\hat{c}_{\vec{j},\sigma}+h.c.\right)\right)^{2}. \tag{156}$$

This model posses a higher O(2N) symmetry and shows no sign problem for all values of N. The model is implemented on the π-flux and square lattices. Tests are included in the test suite git `Testsuite_General_QMCT_code`.

6.1.7 Long range Coulomb

The model we consider here reads:

$$\hat{H}=-t\sum_{\vec{i},\vec{j},\sigma=1}^{N}\hat{c}^{\dagger}_{\vec{i},\sigma}T_{\vec{i},\vec{j}}\hat{c}_{\vec{j},\sigma}+\frac{1}{N}\sum_{\vec{i},\vec{j}}\left(\hat{n}_{\vec{i}}-\frac{N}{2}\right)V_{\vec{i},\vec{j}}\left(\hat{n}_{\vec{j}}-\frac{N}{2}\right)\quad\text{with }\hat{n}_{\vec{i}}=\sum_{\sigma=1}^{N}\hat{c}^{\dagger}_{\vec{i},\sigma}\hat{c}_{\vec{i},\sigma} \tag{157}$$

The interaction is specified in the following way:

$$V_{\vec{i},\vec{j}}=U\begin{cases}1 & \text{if }\vec{i}-\vec{j}=0\\ \frac{\alpha\, d_{min}}{|\vec{i}-\vec{j}|} & \text{otherwise}\end{cases} \tag{158}$$

Here d_{min} is the minimal distance between two orbitals. The code uses the following HS decomposition:

$$e^{-\Delta\tau \hat{H}_V} = \int \prod_{\vec{i}} d\phi_{\vec{i}} e^{-\frac{N\Delta\tau}{4}\phi_{\boldsymbol{i}} V^{-1}_{\boldsymbol{i},\boldsymbol{j}} \phi_{\boldsymbol{j}} - \sum_{\boldsymbol{i}} i\Delta\tau\phi_i\left(n_i - \frac{N}{2}\right)} \tag{159}$$

The implementation follows Ref. [25] but now supports various lattice geometries. The definition of the Coulomb repulsion is as follows. A general lattice site `I,n` where `I: 1...Latt%N` is the unit cell and `n = 1 ...Latt_unit%NORB` the orbital is given by:

```
X_p(:) = Latt%list(I,1)*latt%a1_p(:)  + Latt%list(I,2)*latt%a2_p(:)
         +   Latt_unit%Orb_pos_p(no_j,:)
```

or in more compact notation $\vec{i}+\vec{\delta}_i$. By definition `Latt_unit%Orb_pos_p(1,:)=0`. The Coulomb repulsion between points $\vec{i}+\vec{\delta}_i$ and $\vec{j}+\vec{\delta}_j$ reads:

$$V(\vec{i}+\vec{\delta}_i, \vec{j}+\vec{\delta}_j) = \frac{U d_{min}\alpha}{|\overline{\vec{i}-\vec{j}} + \vec{\delta}_i - \vec{\delta}_j|} \tag{160}$$

Here we use periodic boundary conditions such that $\overline{\vec{i}-\vec{j}}$ is an element of the real space lattice. Note that this is encoded in the array `Latt%imj(I,J)`.

6.2 Model Classes

I suggest the following for the organization of the models. Five independent Hamiltonian files would suffice.

6.2.1 Hubbard models `tU_mod.F90`

$$\sum_{\vec{i},\vec{j},\sigma=1}^{N} \hat{c}^{\dagger}_{\vec{i},\sigma} T_{\vec{i},\vec{j}} \hat{c}_{\vec{i},\sigma} + \frac{U}{N}\sum_{\vec{i}}\left(\sum_{\sigma=1}^{N}\left[\hat{c}^{\dagger}_{\vec{i},\sigma}\hat{c}_{\vec{i},\sigma} - 1/2\right]\right)^2 \tag{161}$$

This Hamiltonian would

- support square, honeycomb, π-flux. It would be very nice to have bilayer versions of these lattices as well.
- Breakdown of the $U(2N)$ symmetry to $U(N)\times U(N)$ so as to accommodate magnetic fields and pinning fields.

6.2.2 Hubbard models `tV_mod.F90`

This would include the $SU(N)$ $t-V$ models on various lattices. The defining property of this set of Hamiltonians would be the enlarged O(2N) symmetry. Again this module should support our standard bipartitie lattices.

6.2.3 Hubbard models `LRC_mod.F90`

This is the long range Coulomb. See above. Again we should include the standard lattices.

6.2.4 Hubbard models `Z2_mod.F90`

I suggest to work on the Hamiltonian of Ref. 6.1.5 since this is the most general model I can think of. Would be nice to add a Hubbard-U term. It is actually not so easy to generalize this model to arbitrary lattices, so that for the moment, I would concentrate only on the square lattice.

6.2.5 Hubbard models `Kondo_mod.F90`

Kondo lattice model on various lattices. I still have to think about the best way of doing things here.

6.3 Performance, memory requirements and parallelization

As mentioned in the introduction, the auxiliary field QMC algorithm scales linearly in inverse temperature β and cubic in the volume N_{dim}. Using fast updates, a single spin flip requires $(N_{\mathrm{dim}})^2$ operations to update the Green function upon acceptance. As there are $L_{\mathrm{Trotter}} \times N_{\mathrm{dim}}$ spins to be visited, the total computational cost for one sweep is of the order of $\beta(N_{\mathrm{dim}})^3$. This operation dominates the performance, see Fig. 3. A profiling analysis of our code shows that 80-90% of the CPU time is spend in ZGEMM calls of the BLAS library provided in the MKL package by Intel. Consequently, the single-core performance is next to optimal.

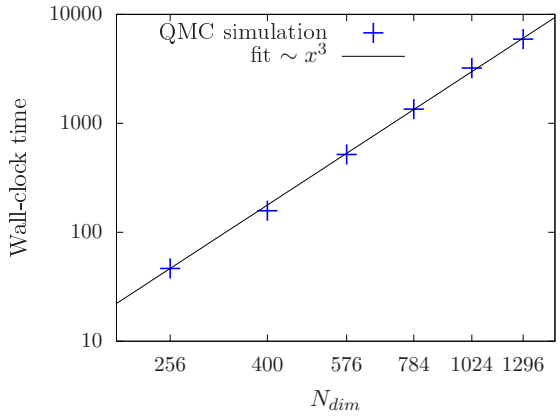

Figure 3: Volume scaling behavior of the auxiliary field QMC code of the ALF project on SuperMUC (phase 2/Haswell nodes) at the LRZ in Munich. The number of sites N_{dim} corresponds to the system volume. The plot confirms that the leading scaling order is due to matrix multiplications such that the runtime is dominated by calls to ZGEMM.

For the implementation which scales linearly in β, one has to store $L_{\mathrm{Trotter}}/$`NWrap` intermediate propagation matrices of dimension $N \times N$. For large lattices and/or low temperatures this dominates the total memory requirements that can exceed 2 GB memory for a sequential version.

At the heart of Monte Carlo schemes lies a random walk through the given configuration space. This is easily parallelized via MPI by associating one random walker to each MPI task. For each task, we start from a random configuration and have to invest the autocorrelation time T_{auto} to produce an equilibrated configuration. Additionally we can also profit from an OpenMP parallelized version of the BLAS/LAPACK library for an additional speedup, which also effects equilibration overhead $N_{\mathrm{MPI}} \times T_{\mathrm{auto}}/N_{\mathrm{OMP}}$, where N_{MPI} is the number of cores and N_{OMP} the number of OpenMP threads. For a given number of independent measurements N_{meas}, we therefore need a wall-clock time given by

$$T = \frac{T_{\mathrm{auto}}}{N_{\mathrm{OMP}}} \left(1 + \frac{N_{\mathrm{meas}}}{N_{\mathrm{MPI}}}\right) . \tag{162}$$

As we typically have $N_{\mathrm{meas}}/N_{\mathrm{MPI}} \gg 1$, the speedup is expected to be almost perfect, in accordance with the performance test results for the auxiliary field QMC code on SuperMUC (see Fig. 4 (left)).

For many problem sizes, 2 GB memory per MPI task (random walker) suffices such that we typically start as many MPI tasks as there are physical cores per node. Due to the large amount of CPU time spent in MKL routines, we do not profit from the hyper-threading option. For large systems, the memory requirement increases and this is tackled by increasing the amount of OpenMP threads to decrease the stress on the memory system and to simultaneously reduce the equilibration overhead (see Fig. 4 (right)). For the displayed speedup, it was crucial to pin the MPI tasks as well as the OpenMP threads in a pattern which keeps the threads as compact as possible to profit from a shared cache. This also explains the drop in efficiency from 14 to 28 threads where the OpenMP threads are spread over both sockets.

We store the field configurations of the random walker as checkpoints, such that a long simulation can be easily split into several short simulations. This procedure allows us to take advantage of chained jobs using the dependency chains provided by the batch system.

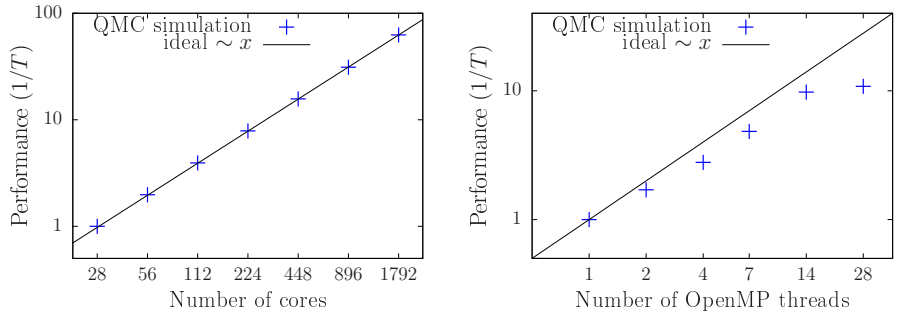

Figure 4: MPI (left) and OpenMP (right) scaling behavior of the auxiliary field QMC code of the ALF project on SuperMUC (phase 2/Haswell nodes) at the LRZ in Munich. The MPI performance data was normalized to 28 cores and was obtained using a problem size of $N_{\mathrm{dim}} = 400$. This is a medium to small system size that is the least favorable in terms of MPI synchronization effects. The OpenMP performance data was obtained using a problem size of $N_{\mathrm{dim}} = 1296$. Employing 2 and 4 OpenMP threads introduces some synchronization/management overhead such that the per-core performance is slightly reduced, compared to the single thread efficiency. Further increasing the amount of threads to 7 and 14 keeps the efficiency constant. The drop in performance of the 28 thread configuration is due to the architecture as the threads are now spread over both sockets of the node. To obtain the above results, it was crucial to pin the processes in a fashion that keeps the OpenMP threads as compact as possible.

7 Conclusions and Future Directions

In its present form, the auxiliary field QMC code of the ALF project allows to simulate a large class of non-trivial models, both efficiently and at minimal programming cost. There are many possible extensions which deserve to be considered in future releases. The model Hamiltonians we have presented so far are imaginary-time independent. This however can be easily generalized to imaginary-time dependent model Hamiltonians thus allowing, for example, to access entanglement properties of interacting fermionic systems [30, 85, 31, 32]. Generalizations to include global moves are equally desirable. This is a prerequisite to play with recent ideas of self-learning algorithms [86] so as to possibly avoid the issue of critical slowing down. At present, the QMC code of this package is restricted to discrete HS fields such that implementations of the long-range Coulomb repulsion – as introduced in [25, 87, 88] – are not yet included. Extensions to continuous HS fields are certainly possible, but require an efficient upgrading scheme. Finally, an implementation of the ground state projective QMC method is equally desirable.

Acknowledgments

We are very grateful to S. Beyl, M. Hohenadler, F. Parisen Toldin, M. Raczkowski, T. Sato, J. Schwab, Z. Wang, and M. Weber for constant support during the development of this project. MB: I think we should acknowledge also RRZE Erlangen: We equally thank G. Hager, M. Wittmann, and G. Wellein for useful discussions and support. FFA would also like to thank T. Lang and Z. Y. Meng for developments of the auxiliary field code as well as T. Grover. MB thanks the Bavarian Competence Network for Technical and Scientific High Performance Computing (KONWIHR) for financial support. FG and JH thank the SFB-1170 for financial support under projects Z03 and C01. FFA thanks the DFG-funded FOR1807 and FOR1346 for partial financial support. Part of the optimization of the code was carried out during the Porting and Tuning Workshop 2016 offered by the Forschungszentrum Jülich. Calculations to extensively test this package were carried out both on SuperMUC at the Leibniz Supercomputing Centre and on JURECA [89] at the Jülich Supercomputing Centre. We thank both institutions for generous allocation of computing time.

References

[1] R. Blankenbecler, D. J. Scalapino, and R. L. Sugar, Phys. Rev. D **24**, 2278 (1981).

[2] S. White, D. Scalapino, R. Sugar, E. Loh, J. Gubernatis, and R. Scalettar, Phys. Rev. B **40**, 506 (1989).

[3] G. Sugiyama and S. Koonin, Annals of Physics **168**, 1 (1986).

[4] S. Sorella, S. Baroni, R. Car, and M. Parrinello, EPL (Europhysics Letters) **8**, 663 (1989).

[5] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, Phys. Lett. **B195**, 216 (1987).

[6] F. Assaad and H. Evertz, in *Computational Many-Particle Physics*, Vol. 739 of *Lecture Notes in Physics*, edited by H. Fehske, R. Schneider, and A. Weiße (Springer, Berlin Heidelberg, 2008), pp. 277–356.

[7] D. J. Scalapino, in *Handbook of High-Temperature Superconductivity: Theory and Experiment*, edited by J. R. Schrieffer and J. S. Brooks (Springer New York, New York, NY, 2007), pp. 495–526.

[8] J. P. F. LeBlanc, A. E. Antipov, F. Becca, I. W. Bulik, G. K.-L. Chan, C.-M. Chung, Y. Deng, M. Ferrero, T. M. Henderson, C. A. Jiménez-Hoyos, E. Kozik, X.-W. Liu, A. J. Millis, N. V. Prokof'ev, M. Qin, G. E. Scuseria, H. Shi, B. V. Svistunov, L. F. Tocchio, I. S. Tupitsyn, S. R. White, S. Zhang, B.-X. Zheng, Z. Zhu, and E. Gull, Phys. Rev. X **5**, 041041 (2015).

[9] M. Hohenadler, T. C. Lang, and F. F. Assaad, Phys. Rev. Lett. **106**, 100403 (2011).

[10] D. Zheng, G.-M. Zhang, and C. Wu, Phys. Rev. B **84**, 205121 (2011).

[11] F. F. Assaad and I. F. Herbut, Phys. Rev. X **3**, 031010 (2013).

[12] F. Parisen Toldin, M. Hohenadler, F. F. Assaad, and I. F. Herbut, Phys. Rev. B **91**, 165108 (2015).

[13] Y. Otsuka, S. Yunoki, and S. Sorella, Phys. Rev. X **6**, 011029 (2016).

[14] S. Chandrasekharan and A. Li, Phys. Rev. D **88**, 021701 (2013).

[15] V. Ayyar and S. Chandrasekharan, Phys. Rev. D **91**, 065035 (2015).

[16] Z.-X. Li, Y.-F. Jiang, S.-K. Jian, and H. Yao, ArXiv:1512.07908 (2015).

[17] F. F. Assaad and T. Grover, Phys. Rev. X **6**, 041049 (2016).

[18] F. F. Assaad, Phys. Rev. Lett. **83**, 796 (1999).

[19] S. Capponi and F. F. Assaad, Phys. Rev. B **63**, 155114 (2001).

[20] Y. Schattner, S. Lederer, S. A. Kivelson, and E. Berg, Phys. Rev. X **6**, 031028 (2016).

[21] E. Berg, M. A. Metlitski, and S. Sachdev, Science **338**, 1606 (2012).

[22] H.-K. Tang, X. Yang, J. Sun, and H.-Q. Lin, Europhys. Lett. **107**, 40003 (2014).

[23] F. F. Assaad, Phys. Rev. B **71**, 075103 (2005).

[24] T. C. Lang, Z. Y. Meng, A. Muramatsu, S. Wessel, and F. F. Assaad, Phys. Rev. Lett. **111**, 066401 (2013).

[25] M. Hohenadler, F. Parisen Toldin, I. F. Herbut, and F. F. Assaad, Phys. Rev. B **90**, 085146 (2014).

[26] H.-K. Tang, E. Laksono, J. N. B. Rodrigues, P. Sengupta, F. F. Assaad, and S. Adam, Phys. Rev. Lett. **115**, 186602 (2015).

[27] M. Rigol, A. Muramatsu, G. G. Batrouni, and R. T. Scalettar, Phys. Rev. Lett. **91**, 130403 (2003).

[28] D. Lee, Progress in Particle and Nuclear Physics **63**, 117 (2009).

[29] T. Grover, Phys. Rev. Lett. **111**, 130402 (2013).

[30] P. Broecker and S. Trebst, Journal of Statistical Mechanics: Theory and Experiment **2014**, P08015 (2014).

[31] F. F. Assaad, T. C. Lang, and F. Parisen Toldin, Phys. Rev. B **89**, 125121 (2014).

[32] F. F. Assaad, Phys. Rev. B **91**, 125146 (2015).

[33] C. Wu and S.-C. Zhang, Phys. Rev. B **71**, 155115 (2005).

[34] E. F. Huffman and S. Chandrasekharan, Phys. Rev. B **89**, 111101 (2014).

[35] Z.-X. Li, Y.-F. Jiang, and H. Yao, Phys. Rev. B **91**, 241117 (2015).

[36] Z. C. Wei, C. Wu, Y. Li, S. Zhang, and T. Xiang, Phys. Rev. Lett. **116**, 250601 (2016).

[37] J. Hubbard, Phys. Rev. Lett. **3**, 77 (1959).

[38] M. Troyer and U.-J. Wiese, Phys. Rev. Lett. **94**, 170201 (2005).

[39] S. Duane and J. B. Kogut, Phys. Rev. Lett. **55**, 2774 (1985).

[40] J. Hirsch, Phys. Rev. B **28**, 4059 (1983).

[41] A. D. Sokal, Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms, 1989, lecture notes from Cours de Troisième Cycle de la Physique en Suisse Romande. Updated in 1996 for the Cargèse Summer School on "Functional Integration: Basics and Applications".

[42] H. G. Evertz, G. Lana, and M. Marcu, Phys. Rev. Lett. **70**, 875 (1993).

[43] A. W. Sandvik, Phys. Rev. B **59**, R14157 (1999).

[44] O. Syljuåsen and A. Sandvik, Phys. Rev. E **66**, 046701 (2002).

[45] J. E. Hirsch and R. M. Fye, Phys. Rev. Lett. **56**, 2521 (1986).

[46] E. Gull, A. J. Millis, A. I. Lichtenstein, A. N. Rubtsov, M. Troyer, and P. Werner, Rev. Mod. Phys. **83**, 349 (2011).

[47] F. F. Assaad, in *DMFT at 25: Infinite Dimensions: Lecture Notes of the Autumn School on Correlated Electrons*, edited by E. Pavarini, E. Koch, D. Vollhardt, and A. Lichtenstein (Verlag des Forschungszentrum Jülich, Jülich, 2014), Vol. 4, Chap. 7. Continuous-time QMC Solvers for Electronic Systems in Fermionic and Bosonic Baths, iSBN 978-3-89336-953-9.

[48] F. F. Assaad and T. C. Lang, Phys. Rev. B **76**, 035116 (2007).

[49] R. T. Scalettar, D. J. Scalapino, and R. L. Sugar, Phys. Rev. B **34**, 7911 (1986).

[50] S. Dürr, Z. Fodor, J. Frison, C. Hoelbling, R. Hoffmann, S. D. Katz, S. Krieg, T. Kurth, L. Lellouch, T. Lippert, K. K. Szabo, and G. Vulvert, Science **322**, 1224 (2008).

[51] F. F. Assaad, in *Lecture notes of the Winter School on Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms.*, edited by J. Grotendorst, D. Marx, and A. Muramatsu. (Publication Series of the John von Neumann Institute for Computing, Jülich, 2002), Vol. 10, pp. 99–155.

[52] Y. Motome and M. Imada, Journal of the Physical Society of Japan **66**, 1872 (1997).

[53] F. F. Assaad, M. Imada, and D. J. Scalapino, Phys. Rev. B **56**, 15001 (1997).

[54] R. M. Fye, Phys. Rev. B **33**, 6271 (1986).

[55] M. Iazzi and M. Troyer, Phys. Rev. B **91**, 241118 (2015).

[56] S. M. A. Rombouts, K. Heyde, and N. Jachowicz, Phys. Rev. Lett. **82**, 4155 (1999).

[57] E. Gull, P. Werner, O. Parcollet, and M. Troyer, EPL (Europhysics Letters) **82**, 57003 (2008).

[58] S. Rombouts, K. Heyde, and N. Jachowicz, Physics Letters A **242**, 271 (1998).

[59] D. Rost, E. V. Gorelik, F. Assaad, and N. Blümer, Phys. Rev. B **86**, 155109 (2012).

[60] D. Rost, F. Assaad, and N. Blümer, Phys. Rev. E **87**, 053305 (2013).

[61] N. Blümer, ArXiv e-prints (2008).

[62] K. Hukushima and K. Nemoto, Journal of the Physical Society of Japan **65**, 1604 (1996).

[63] C. J. Geyer, in *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface* (American Statistical Association, New York, 1991), pp. 156–163.

[64] C. W. Gardiner, *Handbook of Stochastic Methods* (Springer, ADDRESS, 1985).

[65] Z. Bai, C. Lee, R.-C. Li, and S. Xu, Linear Algebra and its Applications **435**, 659 (2011).

[66] A. van der Sluis, Numerische Mathematik **14**, 14 (1969).

[67] J. W. Negele and H. Orland, *Quantum Many body systems*, *Frontiers in physics* (Addison-Wesley, Redwood City, Calif. u.a., 1988).

[68] W. Krauth, *Statistical Mechanics: Algorithms and Computations* (Oxford University Press, ADDRESS, 2006).

[69] C. J. Geyer, Statistical Science **7**, 473 (1992).

[70] R. M. Neal, (1993).

[71] M. Bercx, J. S. Hofmann, F. F. Assaad, and T. C. Lang, Phys. Rev. B **95**, 035108 (2017).

[72] B. Efron and C. Stein, Ann. Statist. **9**, 586 (1981).

[73] S. Chakravarty, B. I. Halperin, and D. R. Nelson, Phys. Rev. Lett. **60**, 1057 (1988).

[74] M. B. Thompson, ArXiv e-prints (2010).

[75] K. S. D. Beach, eprint arXiv:cond-mat/0403055 (2004).

[76] I. Milat, F. Assaad, and M. Sigrist, Eur. Phys. J. B **38**, 571 (2004), http://xxx.lanl.gov/cond-mat/0312450.

[77] M. Bercx, T. C. Lang, and F. F. Assaad, Phys. Rev. B **80**, 045412 (2009).

[78] X. Y. Xu, K. S. D. Beach, K. Sun, F. F. Assaad, and Z. Y. Meng, ArXiv:1602.07150 (2016).

[79] K. S. D. Beach, P. A. Lee, and P. Monthoux, Phys. Rev. Lett. **92**, 026401 (2004).

[80] T. Sato, F. F. Assaad, and T. Grover, Phys. Rev. Lett. **120**, 107201 (2018).

[81] Z.-X. Li, Y.-F. Jiang, and H. Yao, New Journal of Physics **17**, 085003 (2015).

[82] A. Rüegg, S. D. Huber, and M. Sigrist, Phys. Rev. B **81**, 155118 (2010).

[83] M. Hohenadler and F. F. Assaad, arXiv:1906.11937 arXiv:1906.11937 (2019).

[84] S. Gazit, F. F. Assaad, and S. Sachdev, arXiv:1906.11250 arXiv:1906.11250 (2019).

[85] F. F. Assaad, Nat Phys **10**, 905 (2014).

[86] X. Y. Xu, Y. Qi, J. Liu, L. Fu, and Z. Y. Meng, arXiv:1612.03804 (2016).

[87] M. V. Ulybyshev, P. V. Buividovich, M. I. Katsnelson, and M. I. Polikarpov, Phys. Rev. Lett. **111**, 056801 (2013).

[88] R. Brower, C. Rebbi, and D. Schaich, PoS(Lattice 2011)056 (arXiv:1204.5424) .

[89] Jülich Supercomputing Centre, Journal of large-scale research facilities **2**, A62 (2016).

License

The ALF code is provided as an open source software such that it is available to all and we hope that it will be useful. If you benefit from this code we ask that you acknowledge the ALF collaboration as mentioned on our homepage alf.physik.uni-wuerzburg.de. The git repository at alf.physik.uni-wuerzburg.de gives us the tools to create a small but vibrant community around the code and provides a suitable entry point for future contributors and future developments. The homepage is also the place where the original source files can be found. With the coming public release it was necessary to add copyright headers to our source files. The Creative Commons licenses are a good way to share our documentation and it is also well accepted by publishers. Therefore this documentation is licensed to you under a CC-BY-SA license. This means you can share it and redistribute it as long as you cite the original source and license your changes under the same license. The details are in the file license.CCBYSA that you should have received with this documentation. The source code itself is licensed under a GPL license to keep the source as well as any future work in the community. To express our desire for a proper attribution we decided to make this a visible part of the license. To that end we have exercised the rights of section 7 of GPL version 3 and have amended the license terms with an additional paragraph that expresses our wish that if an author has benefitted from this code that he/she should consider giving back a citation as specified on alf.physik.uni-wuerzburg.de. This is not something that is meant to restrict your freedom of use, but something that we strongly expect to be good scientific conduct. The original GPL license can be found in the file license.GPL and the additional terms can be found in license.additional. In favour to our users, the ALF code contains part of the lapack implementation version 3.6.1 from http://www.netlib.org/lapack. Lapack is licensed under the modified BSD license whose full text can be found in license.BSD.
With that being said, we hope that the ALF code will prove to you to be a suitable and high-performance tool that enables you to perform quantum Monte Carlo studies of solid state models of unprecedented complexity.

The ALF project's contributors.

COPYRIGHT