The *ALF* (*Al*gorithms for *L*attice *F*ermions) project release 0.5

Documentation for the auxiliary field quantum Monte Carlo code.

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

March 2, 2017

Contents

1 Introduction

1.1 Motivation

The auxiliary field quantum Monte Carlo (QMC) approach is the algorithm of choice to simulate a variety of correlated electron systems in the solid state and beyond [1, 2]. The phenomena one can investigate in detail include correlation effects in the bulk and surfaces of topological insulators, quantum phase transitions between semimetals (Dirac fermions) and insulators, deconfined quantum critical points, topologically ordered phases, heavy fermion systems, nematic and magnetic quantum phase transitions in metals, superconductivity in spin-orbit split bands, SU(N) symmetric models, etc. This ever growing list of phenomena is based on recent symmetry related insights enabling one to find sign-free formulations of the problem thus allowing for solutions in polynomial time [3, 4]. The aim of the ALF project is to introduce a general formulation of the finite temperature auxiliary field QMC method so as to quickly be able to play with different model Hamiltonians at minimal programming cost. The reader is expected to be somewhat familiar with the auxiliary field QMC approach. A detailed review containing all the prerequisites for understanding the code can be found in [2]. In this documentation, we briefly list the most important equations of the auxiliary field QMC method and then 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.

1.2 Definition of the Hamiltonian

The first and most important part 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 one-body term coupled to an Ising field with dynamics to be specified by the user. The form of the interaction in terms of sums of perfect squares allows us to use generic forms of discrete approximations to the Hubbard-Stratonovich (HS) transformation. Symmetry considerations are imperative to enhance 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:

$$\begin{aligned}
\hat{\mathcal{H}} &= \hat{\mathcal{H}}_T + \hat{\mathcal{H}}_V + \hat{\mathcal{H}}_I + \hat{\mathcal{H}}_{0,I} \text{ , where} \qquad (1)\\
\hat{\mathcal{H}}_T &= \sum_{k=1}^{M_T}\sum_{\sigma=1}^{N_{\mathrm{col}}}\sum_{s=1}^{N_{\mathrm{fl}}}\sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}^{\dagger}_{x\sigma s} T^{(ks)}_{xy} \hat{c}_{y\sigma s} \equiv \sum_{k=1}^{M_T} \hat{T}^{(k)} \qquad (2)\\
\hat{\mathcal{H}}_V &= \sum_{k=1}^{M_V} U_k \left\{ \sum_{\sigma=1}^{N_{\mathrm{col}}}\sum_{s=1}^{N_{\mathrm{fl}}} \left[\left(\sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}^{\dagger}_{x\sigma s} V^{(ks)}_{xy} \hat{c}_{y\sigma s} \right) + \alpha_{ks} \right] \right\}^2 \equiv \sum_{k=1}^{M_V} U_k \left(\hat{V}^{(k)} \right)^2 \qquad (3)\\
\hat{\mathcal{H}}_I &= \sum_{k=1}^{M_I} \hat{Z}_k \left(\sum_{\sigma=1}^{N_{\mathrm{col}}}\sum_{s=1}^{N_{\mathrm{fl}}}\sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}^{\dagger}_{x\sigma s} I^{(ks)}_{xy} \hat{c}_{y\sigma s} \right) \equiv \sum_{k=1}^{M_I} \hat{Z}_k \hat{I}^{(k)} \;. \qquad (4)
\end{aligned}$$

The indices have the following meaning:

- The number of fermion *flavors* is set by N_{fl}. After the Hubbard-Stratonovich transformation, the action will be block diagonal in the flavor index.
- The number of fermion *colors* is set by N_{col}. The Hamiltonian is invariant under SU(N_{col}) rotations. Note that in the code $N_{\mathrm{col}} \equiv$ `N_SUN`.
- The indices x, y label lattice sites where $x, y = 1, \cdots, N_{\mathrm{dim}}$.

  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.
- 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 labelled by M_V and M_I. $M_T > 1$ would allow for a checkerboard decomposition.

The Ising part of the general Hamiltonian (1) is $\hat{\mathcal{H}}_{0,I} + \hat{\mathcal{H}}_I$ and has the following properties:

- $\hat{Z}_k$ is an Ising spin operator which corresponds to the Pauli matrix $\hat{\sigma}_z$. It couples to a general one-body term.
- 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. 4.4 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_{col}) symmetric. We also require the matrices $\boldsymbol{T}^{(ks)}$, $\boldsymbol{V}^{(ks)}$ and $\boldsymbol{I}^{(ks)}$ to be Hermitian.

1.3 Outline

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 equally discuss the Monte Carlo updates as well as the strategies for numerical stabilization of the code.

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 site-dependent equal-time and time-displaced observables, as well as scalar observables.

The Monte Carlo run and the data analysis are separated: 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 three observable types. The analysis programs allow for omitting a given number of initial bins in order to account for warmup times. Also, a rebinning analysis is included to a posteriori take account of long autocorrelation times. Finally, Sec. 3.6 provides all the necessary details to compile and run the code.

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. 4.1, 4.2 and 4.3) as well as for the Hubbard model on a square lattice coupled to a transverse Ising field (see Sec. 4.4). Our implementation is rather general such that a variety of other models can be simulated. In Sec. 5 we provide some information on how to simulate the Kondo lattice as well as the SU(N) symmetric Hubbard-Heisenberg model.

Finally, in Sec. 6 we list a number of features that are considered for future releases of the ALF program package.

2 QMC Essentials

2.1 Formulation of the QMC

The formulation of the Monte Carlo simulation is based on the following.

- We will discretize the imaginary time propagation: $\beta = \Delta\tau L_{\mathrm{Trotter}}$
- We will use the discrete Hubbard-Stratonovich transformation:

$$e^{\Delta\tau\lambda\hat{A}^2} = \sum_{l=\pm1,\pm2} \gamma(l) e^{\sqrt{\Delta\tau\lambda}\eta(l)\hat{A}} + \mathcal{O}(\Delta\tau^4) \,, \tag{5}$$

where the fields η and γ take the values:

$$\begin{aligned} \gamma(\pm1) &= 1+\sqrt{6}/3, \quad \eta(\pm1) = \pm\sqrt{2\left(3-\sqrt{6}\right)} \,, \\ \gamma(\pm2) &= 1-\sqrt{6}/3, \quad \eta(\pm2) = \pm\sqrt{2\left(3+\sqrt{6}\right)} \,. \end{aligned} \tag{6}$$

- We will work in a basis for the Ising spins where $\hat{Z}_k$ is diagonal: $\hat{Z}_k|s_k\rangle = s_k|s_k\rangle$, where $s_k = \pm1$.

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

$$C = \{s_{i,\tau}, l_{j,\tau} \text{ with } i = 1\cdots M_I,\ j = 1\cdots M_V,\ \tau = 1\cdots L_{\text{Trotter}}\} \tag{7}$$

Here, the Ising spins take the values $s_{i,\tau} = \pm 1$ and the Hubbard-Stratonovich fields take the values $l_{j,\tau} = \pm 2, \pm 1$.

2.1.1 The partition function

With the above, the partition function of the model (1) can be written as follows.

$$\begin{aligned}
Z &= \text{Tr}\left(e^{-\beta\hat{\mathcal{H}}}\right) \\
&= \text{Tr}\left[e^{-\Delta\tau\hat{\mathcal{H}}_{0,I}}\prod_{k=1}^{M_T} e^{-\Delta\tau\hat{T}^{(k)}}\prod_{k=1}^{M_V} e^{-\Delta\tau U_k\left(\hat{V}^{(k)}\right)^2}\prod_{k=1}^{M_I} e^{-\Delta\tau\hat{\sigma}_k\hat{I}^{(k)}}\right]^{L_{\text{Trotter}}} + \mathcal{O}(\Delta\tau^2) \\
&= \sum_C\left(\prod_{k=1}^{M_V}\prod_{\tau=1}^{L_{\text{Trotter}}}\gamma_{k,\tau}\right)e^{-S_{0,I}(\{s_{i,\tau}\})}\times \\
&\quad \text{Tr}_{\text{F}}\left\{\prod_{\tau=1}^{L_{\text{Trotter}}}\left[\prod_{k=1}^{M_T} e^{-\Delta\tau\hat{T}^{(k)}}\prod_{k=1}^{M_V} e^{\sqrt{-\Delta\tau U_k}\eta_{k,\tau}\hat{V}^{(k)}}\prod_{k=1}^{M_I} e^{-\Delta\tau s_{k,\tau}\hat{I}^{(k)}}\right]\right\} + \mathcal{O}(\Delta\tau^2)\,.
\end{aligned} \tag{8}$$

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. $S_{0,I}\left(\{s_{i,\tau}\}\right)$ 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. 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:

$$\begin{aligned}
&\text{Tr}_{\text{F}}\left\{\prod_{\tau=1}^{L_{\text{Trotter}}}\left[\prod_{k=1}^{M_T} e^{-\Delta\tau\hat{T}^{(k)}}\prod_{k=1}^{M_V} e^{\sqrt{-\Delta\tau U_k}\eta_{k,\tau}\hat{V}^{(k)}}\prod_{k=1}^{M_I} e^{-\Delta\tau s_{k,\tau}\hat{I}^{(k)}}\right]\right\} = \\
&\prod_{s=1}^{N_{\text{fl}}}\left[e^{\sum_{k=1}^{M_V}\sum_{\tau=1}^{L_{\text{Trotter}}}\sqrt{-\Delta\tau U_k}\alpha_{k,s}\eta_{k,\tau}}\right]^{N_{\text{col}}}\times \\
&\prod_{s=1}^{N_{\text{fl}}}\left[\det\left(\mathbb{1}+\prod_{\tau=1}^{L_{\text{Trotter}}}\prod_{k=1}^{M_T} e^{-\Delta\tau\mathbf{T}^{(ks)}}\prod_{k=1}^{M_V} e^{\sqrt{-\Delta\tau U_k}\eta_{k,\tau}\boldsymbol{V}^{(ks)}}\prod_{k=1}^{M_I} e^{-\Delta\tau s_{k,\tau}\boldsymbol{I}^{(ks)}}\right)\right]^{N_{\text{col}}}\,,
\end{aligned} \tag{9}$$

where the matrices $\mathbf{T}^{(ks)}$, $\mathbf{V}^{(ks)}$, and $\mathbf{I}^{(ks)}$ define the Hamiltonian [Eq. (1) - (4)]. All in all, the partition function is given by:

$$\begin{aligned}
Z &= \sum_C e^{-S_{0,I}(\{s_{i,\tau}\})}\left(\prod_{k=1}^{M_V}\prod_{\tau=1}^{L_{\text{Trotter}}}\gamma_{k,\tau}\right)e^{N_{\text{col}}\sum_{s=1}^{N_{\text{fl}}}\sum_{k=1}^{M_V}\sum_{\tau=1}^{L_{\text{Trotter}}}\sqrt{-\Delta\tau U_k}\alpha_{k,s}\eta_{k,\tau}}\times \\
&\quad \prod_{s=1}^{N_{\text{fl}}}\left[\det\left(\mathbb{1}+\prod_{\tau=1}^{L_{\text{Trotter}}}\prod_{k=1}^{M_T} e^{-\Delta\tau\boldsymbol{T}^{(ks)}}\prod_{k=1}^{M_V} e^{\sqrt{-\Delta\tau U_k}\eta_{k,\tau}\boldsymbol{V}^{(ks)}}\prod_{k=1}^{M_I} e^{-\Delta\tau s_{k,\tau}\boldsymbol{I}^{(ks)}}\right)\right]^{N_{\text{col}}} \\
&\equiv \sum_C e^{-S(C)}\,.
\end{aligned} \tag{10}$$

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 $SU(N_{\text{col}})$ 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:

$$\langle \hat{O} \rangle = \frac{\mathrm{Tr}\left[e^{-\beta \hat{H}} \hat{O} \right]}{\mathrm{Tr}\left[e^{-\beta \hat{H}} \right]} = \sum_C P(C) \langle\langle \hat{O} \rangle\rangle_{(C)}, \text{ with } P(C) = \frac{e^{-S(C)}}{\sum_C e^{-S(C)}} , \tag{11}$$

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 Wicks theorem to compute $O(C)$ from the knowledge of the single particle Green function:

$$G(x, \sigma, s, \tau | x', \sigma', s', \tau') = \langle\langle \mathcal{T} \hat{c}_{x\sigma s}(\tau) \hat{c}^{\dagger}_{x'\sigma' s'}(\tau') \rangle\rangle_C \tag{12}$$

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

$$G(x, \sigma, s, \tau | x', \sigma', s', \tau) = \langle\langle \mathcal{T} \hat{c}_{x\sigma s}(\tau) \hat{c}^{\dagger}_{x'\sigma' s'}(\tau) \rangle\rangle_C \tag{13}$$

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 on the color index. The later reflects the explicit SU(N) color symmetry present at the level of individual HS configurations.

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

$$\begin{aligned} &\langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_1}(\tau_1) c_{\underline{x}'_1}(\tau'_1) \cdots c^{\dagger}_{\underline{x}_n}(\tau_n) c_{\underline{x}'_n}(\tau'_n) \rangle\rangle_C = \\ &\det \begin{bmatrix} \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_1}(\tau_1) c_{\underline{x}'_1}(\tau'_1) \rangle\rangle_C & \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_1}(\tau_1) c_{\underline{x}'_2}(\tau'_2) \rangle\rangle_C & \dots & \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_1}(\tau_1) c_{\underline{x}'_n}(\tau'_n) \rangle\rangle_C \\ \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_2}(\tau_2) c_{\underline{x}'_1}(\tau'_1) \rangle\rangle_C & \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_2}(\tau_2) c_{\underline{x}'_2}(\tau'_2) \rangle\rangle_C & \dots & \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_2}(\tau_2) c_{\underline{x}'_n}(\tau'_n) \rangle\rangle_C \\ \vdots & \vdots & \ddots & \vdots \\ \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_n}(\tau_n) c_{\underline{x}'_1}(\tau'_1) \rangle\rangle_C & \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_n}(\tau_n) c_{\underline{x}'_2}(\tau'_2) \rangle\rangle_C & \dots & \langle\langle \mathcal{T} c^{\dagger}_{\underline{x}_n}(\tau_n) c_{\underline{x}'_n}(\tau'_n) \rangle\rangle_C \end{bmatrix} \end{aligned} \tag{14}$$

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 function. Using the above formulation of Wicks 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. 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:

$$Z = \sum_C e^{-S(C)} = \sum_C \overline{e^{-S(C)}} = \sum_C \Re \left[e^{-S(C)} \right] . \tag{15}$$

Thereby[1] and with the definition

$$\text{sign } (C) = \frac{\Re \left[e^{-S(C)} \right]}{\left| \Re \left[e^{-S(C)} \right] \right|} , \tag{16}$$

[1]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.

the computation of the observable [Eq. (11)] is re-expressed as follows:

$$\begin{aligned}
\langle \hat{O} \rangle &= \frac{\sum_C e^{-S(C)} \langle\langle \hat{O} \rangle\rangle_{(C)}}{\sum_C e^{-S(C)}} \\
&= \frac{\sum_C \Re\left[e^{-S(C)}\right] \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]} \langle\langle \hat{O} \rangle\rangle_{(C)}}{\sum_C \Re\left[e^{-S(C)}\right]} \\
&= \frac{\left\{\sum_C \left|\Re\left[e^{-S(C)}\right]\right| \text{ sign } (C) \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]} \langle\langle \hat{O} \rangle\rangle_{(C)}\right\} / \sum_C \left|\Re\left[e^{-S(C)}\right]\right|}{\left\{\sum_C \left|\Re\left[e^{-S(C)}\right]\right| \text{ sign } (C)\right\} / \sum_C \left|\Re\left[e^{-S(C)}\right]\right|} \\
&= \frac{\left\langle \text{ sign } \frac{e^{-S}}{\Re[e^{-S}]} \langle\langle \hat{O} \rangle\rangle \right\rangle_{\overline{P}}}{\langle \text{sign} \rangle_{\overline{P}}} \,. \qquad (17)
\end{aligned}$$

The average sign is

$$\langle \text{sign} \rangle_{\overline{P}} = \frac{\sum_C \left|\Re\left[e^{-S(C)}\right]\right| \text{ sign } (C)}{\sum_C \left|\Re\left[e^{-S(C)}\right]\right|} \,, \qquad (18)$$

and we have $\langle \text{sign} \rangle_{\overline{P}} \in \mathbb{R}$ per definition. According to Eq. (17) and Eq. (18), the Monte Carlo simulation samples the probability distribution

$$\overline{P}(C) = \frac{\left|\Re\left[e^{-S(C)}\right]\right|}{\sum_C \left|\Re\left[e^{-S(C)}\right]\right|} \,. \qquad (19)$$

2.2 Updating schemes

The program allows for different types of updating schemes. Given a configuration C we propose a new one, C', with probability $T_0(C \to C')$ and accept it according to

$$P(C \to C') = \min\left(1, \frac{T_0(C' \to C) W(C')}{T_0(C \to C') W(C)}\right) \qquad (20)$$

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

Variable	Type	Description
`Propose_S0`	Logical	If true, proposes local moves according to the probability e^{-S_0}
`Global_moves`	Logical	If true, allows for global moves.
`N_Global`	Integer	Number of global moves per sweep of single spin flips.
`TEMPERING`	Compiling option	Requires MPI and runs the code in a parallel tempering mode.

Table 1: Variables required to control the updating scheme

2.2.1 The Default: sequential single spin flips

The default updating scheme is a sequential single spin flip algorithm. Consider the Ising spin $s_{i,\tau}$, we will flip it with probability one 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

$$P(C \to C') = \min\left(1, \frac{W(C')}{W(C)}\right) \qquad (21)$$

It is worth noting that this type of sequential spin flip updating does not satisfy detailed balance but the more fundamental stationarity condition.

2.2.2 Sampling of e^{-S_0}

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

$$T_0(C \to C') = \frac{e^{-S_0(C')}}{e^{-S_0(C')} + e^{-S_0(C)}} = 1 - \frac{1}{1 + e^{-S_0(C')}/e^{-S_0(C)}} \tag{22}$$

Note that the function `S0` in the `Hamitonian_example.f90` module computes precisely the ratio $e^{-S_0(C')}/e^{-S_0(C)}$ so that $T_0(C \to C')$ does not require any further programming. Thereby one will accept the proposed move with the probability:

$$P(C \to C') = \min\left(1, \frac{e^{-S_0(C)} W(C')}{e^{-S_0(C)'} W(C)}\right). \tag{23}$$

With Eq. 10 one sees that 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. This sampling scheme is used if the logical variable `Propose_S0` is switched on.

2.2.3 Global updates

The code equally allows for global updates. The user will have to provide two other functions in the module `Hamiltonian_Examples.f90`.

The subroutine `Global_move(T0_Proposal_ratio,nsigma_old)` proposes a global move. `nsigma_old(M_V+ M_I, Ltrot)` is a two dimensional array containing the full configuration C. On output, the new configuration, C',– determined by the user – is to be stored in the array `nsigma(M_V+ M_I, Ltrot)`. `nsigma(M_V+ M_I, Ltrot)` is a global variable declared in the module, `Hamiltonian`. Equally, on output, the variable `T0_Proposal_ratio` contains the proposal ratio

$$\frac{T_0(C' \to C)}{T_0(C \to C')} \tag{24}$$

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`.

To compute the acceptance rejection ratio, the user will equally have to provide the 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`.

Note that global updates are expensive since they require a complete recalculation of the weight. We thereby allow the user to set a variable `N_Global` that allows to determine how many global updates per sweeps will be carried out.

2.2.4 Parallel Tempering

Exchange Monte Carlo [5] or parallel tempering [6] is a possible route overcome sampling issues in part of 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, 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: $h_1, h_2, h_3 \cdots h_n$, with $h_1 < h_2 < \cdots < h_n$. One will sample the extended ensemble:

$$P([h_1, C_1], [h_2, C_2], \cdots, [h_n, C_n]) = \frac{W(h_1, C_1) W(h_2, C_2) \cdots W(h_n, C_n)}{\sum_{C_1, C_2, \cdots, C_n} W(h_1, C_1) W(h_2, C_2) \cdots W(h_n, C_n)} \tag{25}$$

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:

$$[h_1, C_1], \cdots, [h_i, C_i], [h_{i+1}, C_{i+1}] \cdots, [h_n, C_n] \to [h_1, C_1], \cdots, [h_i, C_{i+1}], [h_{i+1}, C_i] \cdots, [h_n, C_n] \tag{26}$$

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

$$\min\left(1, \frac{W(h_i, C_{i+1})W(h_{i+1}, C_i)}{W(h_i, C_i)W(h_{i+1}, C_{i+1})}\right). \tag{27}$$

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.

The choice of the h_i's 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:

$$\mathcal{P}(h, S) = \sum_C P(h, C)\delta(S(h, C) - S). \tag{28}$$

Acceptance of the exchange step requires the distributions $\mathcal{P}(h, S)$ and $\mathcal{P}(h + \Delta h, S)$ to overlap. For $\langle S\rangle_h < \langle S\rangle_{h+\Delta h}$ one can formulate this requirement as:

$$\langle S\rangle_h + \langle \Delta S\rangle_h \simeq \langle S\rangle_{h+\Delta h} - \langle \Delta S\rangle_{h+\Delta h}, \text{ with } \langle \Delta S\rangle_h = \sqrt{\langle (S - \langle S\rangle_h)^2\rangle_h}. \tag{29}$$

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

$$\Delta h \simeq \frac{2\langle \Delta S\rangle_h}{\partial\langle S\rangle_h/\partial h}. \tag{30}$$

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

$$\Delta h \simeq 2h\frac{\sqrt{C}}{C + h\langle H\rangle_h}, \text{ with } C = h^2\langle (H - \langle H\rangle_h)^2\rangle_h \tag{31}$$

Two comments are in order. i) Let us identify h to 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 such that care has to be taken in the choices of h.

ALF comes with a parallel tempering compiler option which we will discuss in section **??**.

2.3 Stabilization - A Peculiarity of the BSS Algorithm

From (10) it can be seen that for the calculation of the Monte Carlo weight and for the observables a long product of matrix exponentials has to be formed. On top of that we need to be able to extract the single particle Greensfunction from the equation

$$G = \left(1 + \prod_{i=1}^{N_\tau} B(\tau_i, \tau_{i+1})\right)^{-1} \tag{32}$$

To boil this down to more familiar terms from linear algebra we remark that we can recast this problem as the question to the solution of the linear system

$$(1 + \prod_i B_i)x = b. \tag{33}$$

The B_i depend on the system size as well as other physical parameters that can be chosen such that a matrix norm of B_i can have any number. From standard perturbation theory for linear systems it is known that the computed solution $\tilde{x}$ would contain a relative error of

$$\frac{|\tilde{x} - x|}{|x|} = \mathcal{O}\left(\epsilon\kappa(1 + \prod_i B_i)\right). \tag{34}$$

Here ϵ denotes the machine precision which is 2^{-53} for IEEE double precision numbers and $\kappa(M)$ is the condition number of the matrix M. The important fact that makes straight-forward inversion so badly suited is the point that each of the B_i contains exponentially large and small scales as can be seen in

(10). The condition number in the above expression is determined by the product of the B_i and hence can be virtually unbounded in size. This means that the so computed solution $\tilde{x}$ will often contain no correct digits at all. To circumvent that more sophisticated methods have to be employed. ALF is by default employing the strategy of forming a product of QR-decompositions which was proven to be weakly backwards stable in [7]. 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 the $B_i = Q_i \tilde{R}_i$. Now Q_i is a unitary matrix and hence $\kappa(Q_i) = 1$. To get a handle on the condition number of $\tilde{R}_i$ we will form the diagonal matrix $(D_i)_{jj} = |(\tilde{R}_i)_{jj}|$ and rescale $\tilde{R}_i$ accordingly, $\tilde{R}_i = D_i R_i$. This gives the decomposition

$$B_i = Q_i D_i R_i \tag{35}$$

D_i now contains the row norms of the original $\tilde{R}_i$ matrix and hence separates off the total scales of the problem since R_i is now only of modest condition number. So given an initial decomposition of $B_{j-1} = Q_{j-1} D_{j-1} T_{j-1}$ any product of B matrices is formed in the following two steps:

1. Form $M_j = (B_j Q_{j-1}) D_{j-1}$. Note the parentheses.
2. Do a QR decomposition of $M_j = Q_j D_j R_j$.
3. Form the updated R matrices $T_j = R_j T_{j-1}$.

While this provides provides a stable method to calculate the involved matrix product it can be pretty expensive. Therefore the user can specify to skip a certain number of QR Decompositions and perform plain multiplications instead. This is specified in the parameters file by the `NWrap` parameter. `NWrap` $= 1$ corresponds to always performing QR decompositions whereas larger integers give longer intervals where no QR decomposition will be performed. The effectiveness of the stabilization *HAS* to be judged for every simulation from the info file. For most simulations there are two values to look out for:

- `PrecisionGreen`
- `PrecisionPhase`

The Green's 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 precision is between $10^{-10} - 10^{-16}$.

2.4 Caveats in QMC simulations

The default updating scheme consists of local moves which changes (upon acceptance) only one single entry of $L_{\text{Trotter}}(M_I + M_V)$ spins (see Sec. 2.2). It is obvious that a single spin flip does not generate a independent configuration C. Hence it will require at least as many local spin flips as there are spins in the configuration space. This is however only the lower bound as there can be a region in the spin space where the fields are correlated and it requires a larger or even global move to significantly change the configuration to an independent one. One might imagine a ferromagnet due to spontaneous symmetry breaking. All spins are parallel aligned and, let' say, point upwards. The configuration of only down spins is equally justified, but rotating one to the other requires a global operation. Flipping the spins individually one after another generates intermediate states of relative high energy which corresponds to a low probability in the QMC algorithm.

These considerations lead to the definition of the auto-correlation time T_{auto} that characterizes the required time scale to generate an independent configuration or values $\langle\langle \hat{O} \rangle\rangle_C$ 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 at least one T_{auto} to generate relevant configurations before reliable measurements are possible. This phase of the simulation is known as the warm-up. In order to keep the code as flexible as possible (different simulations might have different auto-correlation times), measurements are taken from the very beginning. Instead we provide the parameter `n_skip` for the analysis to ignore the first `n_skip` bins.
- Secondly, our implementation averages over a given amount of measurements (details see ...) before storing the results, known as one bin, on the disk. The following error analysis requires independent bins to generate reliable confidence estimates. If bins are to small (averaged over a period shorter then T_{auto}), the error bars are then typically underestimated. Most of the time, the auto-correlation

time is unknown before the simulation is started, sometime, the compute cluster does not allow single runs long enough to generate appropriately sized bins. Therefore we provide the `N_rebin` parameter that specifies how many bins are combined into a new bin during the error analysis. In general, one should check, that a further increase of the bin size does not change the error estimate any more such that the results have converged.

REMARK: The `N_rebin` variable can be used to control a second issue. The distribution of the Monte Carlo estimates $\langle\langle \hat{O} \rangle\rangle_C$ are unknown. The result in the form (best±error) assumes a Gaussian distribution. Luckily, every original distribution with a finite variance turns into a Gaussian one, once if is folded often enough (central limit theorem). Due to the internal averaging (folding) within on bin, many observables are already quite Gaussian. Otherwise one can increase `N_rebin` further, even if the bins are independent already [8].

- The third caveat concerns time-resolved 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; Eqn. 57) 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

$$Cov_{\tau\tau'} = \left\langle \left(S_{\alpha,\beta}(\vec{k},\tau) - \langle S_{\alpha,\beta}(\vec{k},\tau)\rangle \right) \left(S_{\alpha,\beta}(\vec{k},\tau') - \langle S_{\alpha,\beta}(\vec{k},\tau')\rangle \right) \right\rangle . \qquad (36)$$

An example where this information is the extraction of energy scales by fitting the tail around $\frac{\beta}{2}$ that would otherwise underestimate the uncertainty.

3 Data Structures & 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. 4. 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 $\hat{\mathcal{H}}_{I,0}$.	4.4.2
`Alloc_obs`	Asigns 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 (1). 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 $N_{\text{dim}} \times N_{\text{dim}}$, 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

$$X_{x,y} \begin{cases} \neq 0 & \text{if } x, y \in \{z_1, \cdots z_N\} \\ = 0 & \text{otherwise} \end{cases} \tag{37}$$

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

$$P_{i,x} = \delta_{z_i,x} \ , \tag{38}$$

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

$$\boldsymbol{X} = \boldsymbol{P}^T \boldsymbol{O} \boldsymbol{P} \ , \tag{39}$$

and

$$X_{x,y} = \sum_{i,j}^{N} P_{i,x} O_{i,j} P_{j,y} = \sum_{i,j}^{N} \delta_{z_i,x} O_{ij} \delta_{z_j,y} \ . \tag{40}$$

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

$$P_i = z_i. \tag{41}$$

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

$$e^{\boldsymbol{X}} = e^{\boldsymbol{P}^T \boldsymbol{O} \boldsymbol{P}} = \sum_{n=0}^{\infty} \frac{\left(\boldsymbol{P}^T \boldsymbol{O} \boldsymbol{P}\right)^n}{n!} = \mathbb{1} + \boldsymbol{P}^T \left(e^{\boldsymbol{O}} - \mathbb{1}\right) \boldsymbol{P} \ , \tag{42}$$

since

$$\boldsymbol{P}\boldsymbol{P}^T = \mathbb{1}_{N \times N}. \tag{43}$$

In the code, we define a structure called `Operator` to capture the above. This type `Operator` bundles several components 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	Parameter to set the type of HS transformation (1 = Ising, 2 = discrete HS for perfect-square term)
`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 $\mathbf{O}$

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.

In this section we show how to specify the Hamiltonian (1) 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 (10). For each pair of indices (k, s), these terms have the general form

$$\text{Matrix Exponential} = e^{g\,\phi(\texttt{type})\,\mathbf{X}}\ . \tag{44}$$

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. (3). 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 a `Operator` variable:

- the matrix $\mathbf{X}$ [see Eq. (39)]
- 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 (5) and we store `type=2` which sets $\phi_{k\tau} = \eta_{k\tau}$. 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 (1):

- For the hopping Hamiltonian (2), 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

  $$\left(\sum_{x,y}^{N_{\text{dim}}} \hat{c}^{\dagger}_{x} T^{(ks)}_{xy} \hat{c}_{y}\right) , \tag{45}$$

  where $k = [1, M_T]$ and $s = [1, N_{\text{fl}}]$. To make contact with the general expression (44) 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 (3), 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:

  $$\left[\left(\sum_{x,y}^{N_{\text{dim}}} \hat{c}^{\dagger}_{x} V^{(ks)}_{x,y} \hat{c}_{y}\right) + \alpha_{ks}\right] , \tag{46}$$

  where $k = [1, M_V]$ and $s = [1, N_{\text{fl}}]$. To make contact with the general expression (44) 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 (4), 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:

  $$\left(\sum_{x,y}^{N_{\text{dim}}} \hat{c}^{\dagger}_{x} I^{(ks)}_{xy} \hat{c}_{y}\right) , \tag{47}$$

  where $k = [1, M_I]$ and $s = [1, N_{\text{fl}}]$. To make contact with the general expression (44), 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 (3) and Ising term (4)], 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):

$$\hat{c}_{\vec{i}+\vec{L}_1} = \hat{c}_{\vec{i}+\vec{L}_2} = \hat{c}_{\vec{i}} \tag{48}$$

The function call

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

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

$$\vec{a}_i \cdot \vec{g}_i = 2\pi\delta_{i,j}, \tag{49}$$

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:

$$\begin{bmatrix} \vec{L}_1 \cdot \vec{g}_1 & \vec{L}_1 \cdot \vec{g}_2 \\ \vec{L}_2 \cdot \vec{g}_1 & \vec{L}_2 \cdot \vec{g}_2 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = 2\pi \begin{bmatrix} n \\ m \end{bmatrix} \tag{50}$$

such that

$$\vec{k} = n\vec{b}_1 + m\vec{b}_2 \text{ with } \quad \vec{b}_1 = \frac{2\pi}{(\vec{L}_1 \cdot \vec{g}_1)(\vec{L}_2 \cdot \vec{g}_2) - (\vec{L}_1 \cdot \vec{g}_2)(\vec{L}_2 \cdot \vec{g}_1)} \left[(\vec{L}_2 \cdot \vec{g}_2)\vec{g}_1 - (\vec{L}_2 \cdot \vec{g}_1)\vec{g}_2 \right] \text{ and}$$
$$\vec{b}_2 = \frac{2\pi}{(\vec{L}_1 \cdot \vec{g}_1)(\vec{L}_2 \cdot \vec{g}_2) - (\vec{L}_1 \cdot \vec{g}_2)(\vec{L}_2 \cdot \vec{g}_1)} \left[(\vec{L}_1 \cdot \vec{g}_1)\vec{g}_2 - (\vec{L}_1 \cdot \vec{g}_2)\vec{g}_1 \right] \tag{51}$$

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: $\vec{R}_i = \texttt{list(i,1)}\ \vec{a}_1 + \texttt{list(i,2)}\ \vec{a}_2 \equiv i_1\vec{a}_1 + i_2\vec{a}_2$
`Latt%invlist`	Integer	$\texttt{Invlist}(i_1, i_2) = i$
`Latt%nnlist`	Integer	$j = \texttt{nnlist}(i, n_1, n_2)$, $n_1, n_2 \in [-1, 1]$ $\vec{R}_j = \vec{R}_i + n_1\vec{a}_1 + n_2\vec{a}_2$
`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. 49)
`Latt%b1_p, Latt%b1_p`	Real	k-quantization (See Eq. 51)
`Latt%listk`	Integer	Maps each reciprocal lattice point $k = 1, \cdots, N_{\text{unit cell}}$ to a reciprocal space vector $\vec{k}_k = \texttt{listk(k,1)}\vec{b}_1 + \texttt{listk(k,2)}\vec{b}_2 \equiv k_1\vec{b}_1 + k_2\vec{b}_2$
`Latt%invlistk`	Integer	$\texttt{Invlistk}(k_1, k_2) = k$
`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:

$$S(\vec{k}, :, :, :) = \frac{1}{N_{unit\,cell}} \sum_{\vec{i},\vec{j}} e^{-i\vec{k}\cdot(\vec{i}-\vec{j})} S(\vec{i}-\vec{j}, :, :, :) \tag{52}$$

and `Fourier_K_to_R` the inverse Fourier transform

$$S(\vec{r},:,:,:) = \frac{1}{N_{unit\,cell}} \sum_{\vec{k}\in BZ} e^{i\vec{k}\cdot\vec{r}} S(\vec{k},:,:,:). \tag{53}$$

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

3.2 The observable types `Obser_Vec` and `Obser_Latt`

Our definition of the model includes observables [Eq. (17)]. 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 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 bins 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`:

$$\begin{aligned} \texttt{ZS} &= \text{sign}(C)\,, & (54)\\ \texttt{ZP} &= \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]}\,. & (55)\end{aligned}$$

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

$$\texttt{phase} = \frac{e^{-S(C)}}{\left|e^{-S(C)}\right|}\,, \tag{56}$$

which is provided by the main program. Note that each observable structure also includes the average sign [Eq. (18)].

3.2.1 Scalar observables

This data type is described in Table 5 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 unity, the sign (see Eq. 16) is cumulated in the variable `Obs%Ave_sign`, and the desired the observables (multiplied by the sign and $\frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]}$, see Sec. 2.1.2) are cumulated in the vector `Obs%Obs_vec`. At the end

Variable	Type	Description	Contribution of configuration C
`Obs%N`	Int.	Number of measurements	
`Obs%Ave_sign`	Real	Cumulated sign [Eq. (18)]	sign(C)
`Obs%Obs_vec(:)`	Compl.	Cumulated vector of observables [Eq. (17)]	$\langle\langle \hat{O}(:)\rangle\rangle_C \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]}$ sign (C)
`Obs%File_Vec`	Char.	Name of output file	

Table 5: Components of the `Obser_vec` type. The table lists the data included in a variable `Obs` of type `Obser_vec`.

of the 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.

Variable	Type	Description	Contribution of configuration C
`Obs%N`	Int.	Number of measurements	
`Obs%Ave_sign`	Real	Cumulated sign [Eq. (18)]	$\text{sign}(C)$
`Obs%Obs_latt` $(\vec{i}-\vec{j},\tau,\alpha,\beta)$	Compl.	Cumul. correl. fct. [Eq. (17)]	$\langle\langle \hat{O}_{\vec{i},\alpha}(\tau)\hat{O}_{\vec{j},\beta}\rangle\rangle_C \, \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]}\text{sign}(C)$
`Obs%Obs_latt0`(α)	Compl.	Cumul. expect. value [Eq. (17)]	$\langle\langle \hat{O}_{\vec{i},\alpha}\rangle\rangle_C \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]}$ sign (C)
`Obs%File_Latt`	Char.	Name of output file	

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

3.2.2 Equal time and time displaced correlation functions

This data type (see Table 6) is useful so as to deal with equal time as well as imaginary-time displaced correlation functions of the form:

$$S_{\alpha,\beta}(\vec{k},\tau) = \frac{1}{N_{\text{unit cell}}}\sum_{\vec{i},\vec{j}} e^{-\vec{k}\cdot(\vec{i}-\vec{j})}\left(\langle \hat{O}_{\vec{i},\alpha}(\tau)\hat{O}_{\vec{j},\beta}\rangle - \langle \hat{O}_{\vec{i},\alpha}\rangle\langle \hat{O}_{\vec{j},\beta}\rangle\right). \tag{57}$$

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

$$S^{(\text{corr})}_{\alpha,\beta}(\vec{k},\tau) = \frac{1}{N_{\text{unit cell}}}\sum_{\vec{i},\vec{j}} e^{-i\vec{k}\cdot(\vec{i}-\vec{j})}\langle \hat{O}_{\vec{i},\alpha}(\tau)\hat{O}_{\vec{j},\beta}\rangle\,, \tag{58}$$

$$\begin{aligned} S^{(\text{back})}_{\alpha,\beta}(\vec{k}) &= \frac{1}{N_{\text{unit cell}}}\sum_{\vec{i},\vec{j}} e^{-i\vec{k}\cdot(\vec{i}-\vec{j})}\langle \hat{O}_{\vec{i},\alpha}(\tau)\rangle\langle \hat{O}_{\vec{j},\beta}\rangle \\ &= N_{\text{unit cell}}\,\langle \hat{O}_{\alpha}\rangle\langle \hat{O}_{\beta}\rangle\,\delta(\vec{k})\,, \end{aligned} \tag{59}$$

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

$$\langle \hat{O}_\alpha\rangle \equiv \frac{1}{N_{\text{unit cell}}}\sum_{\vec{i}}\langle \hat{O}_{\vec{i},\alpha}\rangle\,. \tag{60}$$

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 cumulates $\langle\langle \hat{O}_{\vec{i},\alpha}(\tau)\hat{O}_{\vec{j},\beta}\rangle\rangle_C \, \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]}\text{sign}(C)$ in `Obs%Obs_latt(`$\vec{i}-\vec{j},\tau,\alpha,\beta$`)` and $\langle\langle \hat{O}_{\vec{i},\alpha}\rangle\rangle_C \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]}$ 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 name convention. 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

The code package consists of the program directories `Prog/`, `Libraries/` and `Analysis/`. The sample simulations corresponding to the walkthroughs of Sec. 4.1 - 4.4 are included in `Examples/`. The package content is summarized in Table 7.

3.3.1 Input files

The input files are listed in Table 8. 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. 4.1 for a detailed walkthrough):

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 7: Overview of the directories.

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.
`confin_<threadnumber>`	Input files for the HS and Ising configuration, used to continue a simulation.

Table 8: Overview of the input files in `Start/` required for a simulation.

```
 ===============================================================================
!  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.
/
```

3.3.2 Output files

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

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 a MPI parallelized version of the code, the average also extends over the number of MPI threads. The standard output files are listed in Table 9.

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. (60)] contributing to the background part [Eq. (59)] of the correlation function, and the correlated part [Eq. (58)] 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
      $\langle \hat{O}_\alpha \rangle$
  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
              $\langle S^{(corr)}_{\alpha,\beta}(k(i),\tau) \rangle$
           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. (57)] is then calculated by calling the error analysis routine (see Sec. 3.5)

3.3.2.1 The `info` file and stabilization

The finite temperature auxiliary field QMC algorithm is known to be numerically unstable. 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. [9] for the finite temperature algorithm and in Refs. [10, 11] for the zero temperature projective algorithm. As shown in Ref. [2] 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. For example, in the directory `Examples/Hubbard_SU2_Square` an example simulation of the 4×4 Hubbard model at $U/t = 4$ and $\beta t = 10$, the `info` file contains the lines

```
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 [2]. 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 1. Numerical stabilization is delicate and there is no guarantee that it will work for all models. For example switching to a HS field which couples to the z-component of the magnetization will yield (see directory `Examples/Hubbard_Mz_Square`):

```
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 a couple of order of magnitudes less precise than a HS decomposition coupling to the charge. 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 the you switch libraries. 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 their PROGRAMMCONFIGURATION string. To use these routines you need to link against a lapack library that implements at least the lapack-3.4.0 interface. [2]

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` or `-DSTAB1` in the `set_env.sh` file and recompile the code.

3.4 Scripts

3.5 Analysis programs

Here we briefly discuss the analysis programs which read in bins and carry out the error analysis. Error analysis is based on the central limit theorem, which required bins to be statistically independent, and also the existence of a well-defined variance of the distribution. The former will be the case if bins are longer than the autocorrelation time. The latter has to be checked by the user, since in general the distribution variance depends on the model and on the observable. 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`

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

Script	Description	Section
`set_env.sh`	Sets the environment variables for the compiler and the libraries.	3.6
`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 10: Overview of the bash script files.

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 resampling.
`cov_eq.f90`	In combination with the script `analysis.sh`, the bin files with suffix `_eq` are read in, and the corresponding files will 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 4.1.2.1). 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.

Table 11: Overview of analysis programs that are called within the script `analysis.sh`.

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. As listed in Table, 11 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

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 9)

Table 12: Standard input files for the error analysis.

files mentioned in Table 13.

- 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
```

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 refers 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 13: Standard output files of the error analysis.

```
         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. 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.

3.6 Running the code

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

3.6.1 Compilation

The environment variables are defined in the bash script set_env.sh as follows:

```
# Description of PROGRAMMCONFIGURATION:
# -DMPI selects MPI.
# Setting nothing  compiles without mpi.
# -DQRREF selects  a reference implementation of the QR decomposition.
# Setting nothing selects system lapack for the QR decomposition.
# -DSTAB1 selects an alternative stabilization scheme.
# Setting nothing selects the default stabilizatition
PROGRAMMCONFIGURATION=""
f90="gfortran"
export f90
F90OPTFLAGS="-O3"
export F90OPTFLAGS
FL="-c ${F90OPTFLAGS} ${PROGRAMMCONFIGURATION}"
export FL
DIR=`pwd`
export DIR
Libs=${DIR}"/Libraries/"
export Libs
LIB_BLAS_LAPACK="-llapack -lblas"
export LIB_BLAS_LAPACK
```

In the above, the GNU Fortan compiler `gfortran` is set.[3] 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. To compile the libraries, the analysis programs and the quantum Monte Carlo program, the following steps should be executed:

1. Export the environment variables:

   ```
   source set_env.sh
   ```

2. Compile the libraries and the error analysis routines

   ```
   cd Libraries
   make
   cd ..
   cd Analysis
   make
   cd ..
   ```

3. Compile the quantum Monte Carlo code

   ```
   cd Prog
   make
   cd ..
   ```

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. 4, 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

To perform an error analysis (based on the jackknife scheme) of the Monte Carlo bins for all observables run the script `analysis.sh` (see Sec. 3.5).

4 Examples

4.1 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 he/she wishes 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

$$\mathcal{H} = \sum_{\sigma=1}^{2} \sum_{x,y=1}^{N_{\text{unit cell}}} c^{\dagger}_{x\sigma} T_{x,y} c_{y\sigma} + \frac{U}{2} \sum_{x} \left[\sum_{\sigma=1}^{2} \left(c^{\dagger}_{x\sigma} c_{x\sigma} - 1/2 \right) \right]^2 . \tag{61}$$

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

[3] 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.

4.1.1 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.

4.1.1.1 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` $\equiv$ `Latt%N` $\cdot$ `NORB`. Since $N_{\text{unit cell}} =$ `Latt%N`, this equation is a bit confusing.

4.1.1.2 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

4.1.1.3 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 $N_{\mathrm{dim}} \times N_{\mathrm{fl}} = N_{\mathrm{dim}} \times 1$. 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 $1 \times N_{\mathrm{dim}}$ 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.

4.1.2 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 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 he/she wants to compute. Here we will describe how to proceed.

4.1.2.1 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.

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

The equal time green function,

$$\texttt{GR(x,y,}\sigma\texttt{)} = \langle c_{x,\sigma} c^{\dagger}_{y,\sigma} \rangle, \tag{62}$$

the phase factor `phase` [Eq. (56)] and time slice `Ntau` is provided by the main program.

Here, x and y label 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. (16)], `ZP` $= \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]}$ [see Eq. (56)] 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 fluctuations:

$$\langle n_{\vec{i},\alpha} n_{\vec{j},\beta} \rangle - \langle n_{\vec{i},\alpha} \rangle \langle n_{\vec{j},\beta} \rangle \tag{63}$$

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

$$\texttt{GRC(x,y,s)} = \delta_{x,y} - \texttt{GR(y,x,s)} \tag{64}$$

such that `GRC(x,y,s)` corresponds to $\langle\langle \hat{c}^{\dagger}_{x,s} \hat{c}_{y,s} \rangle\rangle$.

```
Obs_eq(4)%N        = Obs_eq(4)%N + 1         ! Even if it is redundant, each observable carries
Obs_eq(4)%Ave_sign = Obs_eq(4)%Ave_sign + Real(ZS,kind(0.d0))   ! its own counter and sign.
Do I1 = 1,Ndim
   I    = List(I1,1)                         ! = I  For the Hubbard model  on the square
   no_I = List(I1,2)                         ! = 1  lattice there is 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
```

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.

4.1.2.3 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:

$$\begin{aligned}
\texttt{GT0(x,y,s)} &= \langle\langle \hat{c}_{x,s}(Nt\Delta\tau)\hat{c}^\dagger_{y,s}(0)\rangle\rangle = \langle\langle \mathcal{T}\hat{c}_{x,s}(Nt\Delta\tau)\hat{c}^\dagger_{y,s}(0)\rangle\rangle \\
\texttt{G0T(x,y,s)} &= -\langle\langle \hat{c}^\dagger_{y,s}(Nt\Delta\tau)\hat{c}_{x,s}(0)\rangle\rangle = \langle\langle \mathcal{T}\hat{c}_{x,s}(0)\hat{c}^\dagger_{y,s}(Nt\Delta\tau)\rangle\rangle \\
\texttt{G00(x,y,s)} &= \langle\langle \hat{c}_{x,s}(0)\hat{c}^\dagger_{y,s}(0)\rangle\rangle \\
\texttt{GTT(x,y,s)} &= \langle\langle \hat{c}_{x,s}(Nt\Delta\tau)\hat{c}^\dagger_{y,s}(Nt\Delta\tau)\rangle\rangle
\end{aligned} \tag{65}$$

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:

$$4\langle\langle \hat{S}^z_{\vec{i}}(\tau)\hat{S}^z_{\vec{j}}(0)\rangle\rangle = -2\,\texttt{G0T(J1,I1,1) GT0(I1,J1,1)} \tag{66}$$

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.

4.2 The M_z-Hubbard model on a square lattice

The Hubbard Hamiltonian can equally be written as:

$$\mathcal{H} = \sum_{\sigma=1}^{2}\sum_{x,y=1}^{N_{unit\ cells}} c^\dagger_{x\sigma}T_{x,y}c_{y\sigma} - \frac{U}{2}\sum_x\left[c^\dagger_{x,\uparrow}c_{x\uparrow} - c^\dagger_{x,\downarrow}c_{x\downarrow}\right]^2 . \tag{67}$$

We can make contact with the general form of the Hamiltonian (see Eq. 1) by setting: $N_{\mathrm{fl}} = 2$, $N_{\mathrm{col}} \equiv$ 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 to the z-component of the magnetization breaks the SU(2) spin symmetry. Nevertheless the z-component of 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 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 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 [3, 12, 13]. A sample run for this model can be found in `Examples/Hubbard_Mz_Square/`.

4.2.1 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.

4.2.2 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, $4\langle\langle \hat{S}^z_{\vec{i}} \hat{S}^z_{\vec{j}} \rangle\rangle$, calculated in the subroutine `Obser`, will take the form:

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

Here, `GRC` is defined in Eq. 64. 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.

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

The Hamilton module `Hamiltonian_Examples.f90` can also carry out simulations for the the Hubbard model on the Honeycomb lattice by setting in the parameter file (see Sec. 3.3.1) `Lattice_type = "Honeycomb"`. A sample run for this model can be found in `Examples/Hubbard_SU2_Honeycomb/`.

4.3.1 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 implement this in the code. The Honeycomb lattice is a triangular Bravais lattice the 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 inde nc
      Invlist(I,no) = nc   ! Super index for given  unit cell and orbital
  Enddo
Enddo
```

With the above lists one can run trough 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 on unit cell, and the number of orbitals will then correspond to the number of lattice sites.

4.3.2 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 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_1
      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.

4.3.3 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 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.

4.4 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.

$$\mathcal{H} = \sum_{\sigma=1}^{2}\sum_{x,y} c^{\dagger}_{x\sigma} T_{x,y} c_{y\sigma} + \frac{U}{2}\sum_{x}\left[\sum_{\sigma=1}^{2}\left(c^{\dagger}_{x\sigma}c_{x\sigma} - 1/2\right)\right]^2 + \xi \sum_{\sigma,\langle x,y\rangle} \hat{Z}_{\langle x,y\rangle}\left(c^{\dagger}_{x\sigma}c_{y\sigma} + h.c.\right)$$
$$-h\sum_{\langle x,y\rangle}\hat{X}_{\langle x,y\rangle} - J\sum_{\langle\langle x,y\rangle\langle x',y'\rangle\rangle}\hat{Z}_{\langle x,y\rangle}\hat{Z}_{\langle x',y'\rangle} \qquad (68)$$

We can make contact with the general form of the Hamiltonian by setting: $N_{\text{fl}} = 1$, $N_{\text{col}} \equiv$ `N_SUN` $= 2$, $M_T = 1$, $T^{(ks)}_{xy} = T_{x,y}$, $M_V = N_{\text{unit cell}} \equiv N_{\text{dim}}$, $U_k = -\frac{U}{2}$, $V^{(ks)}_{xy} = \delta_{x,y}\delta_{x,k}$, $\alpha_{ks} = -\frac{1}{2}$ and $M_I = 2N_{\text{unit cell}}$. The last two terms of the above Hamiltonian describes a transverse Ising field model on the bonds of the square lattice. This type of Hamiltonian has recently been extensively discussed [14, 15, 16]. Here we adopt the notation of Ref. [16]. 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.

Firstly, one has to specify the function `Real (Kind=8) function S0(n,nt)` and secondly modify the interaction `Call Ham_V`.

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

4.4.1 The interaction term: `Call Ham_V`

The dimension of `Op_V` is now $(M_I + M_V) \times N_{\text{fl}} = (3 * N_{\text{dim}}) \times 1$. We set the effective dimension for the Hubbard term to $N = 1$ and to $N = 2$ for the Ising term. The allocation of this array of operators reads:

```
do i  = 1,N_coord*Ndim    !  Runs  over bonds for Ising variable
  call Op_make(Op_V(i,1),2)
enddo
do i  =  N_coord*Ndim+1, (N_coord+1)*Ndim   ! Runs over sites for Hubbard
  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) ! Site one 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)   ! Site two 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 first `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
```

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

As mentioned above, a configuration is given by

$$C = \{s_{i,\tau}, l_{j,\tau} \text{ with } i = 1 \cdots M_I, j = 1 \cdots M_V, \tau = 1, L_{Trotter}\} \tag{69}$$

and 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`, `n`, corresponds to the index of the operator string `Op_V(n,1)`. If `Op_V(n,1)%type = 2`, `S0(n,nt)` returns 1. If `Op_V(n,1)%type = 1` then function `S0` returns

$$\frac{e^{-S_{0,I}\left(s_{1,\tau},\cdots,-s_{m,\tau},\cdots s_{M_I,\tau}\right)}}{e^{-S_{0,I}\left(s_{1,\tau},\cdots,s_{m,\tau},\cdots s_{M_I,\tau}\right)}} \tag{70}$$

That is, `S0(n,nt)` returns the ratio of the new to old weight of the Ising Hamiltonian upon flipping a single Ising spin $s_{m,\tau}$. Note that in this specific case `m = n`

5 Miscellaneous

5.1 Other models

The aim of this section is to briefly mention a small selection of other models that can be simulated within the ALF-project.

5.1.1 The Kondo lattice

Simulating the Kondo lattice within the ALF-project requires rewriting of the model along the lines of Refs. [17, 18, 19]. Adopting the notation of these articles, the Hamiltonian that one will simulate reads:

$$\hat{\mathcal{H}} = \underbrace{-t \sum_{\langle \vec{i},\vec{j}\rangle,\sigma} \left(\hat{c}^{\dagger}_{\vec{i},\sigma} \hat{c}_{\vec{j},\sigma} + \text{H.c.} \right)}_{\equiv \hat{\mathcal{H}}_t} - \frac{J}{4} \sum_{\vec{i}} \left(\sum_{\sigma} \hat{c}^{\dagger}_{\vec{i},\sigma} \hat{f}_{\vec{i},\sigma} + \hat{f}^{\dagger}_{\vec{i},\sigma} \hat{c}_{\vec{i},\sigma} \right)^2 + \underbrace{\frac{U}{2} \sum_{\vec{i}} \left(\hat{n}^{f}_{\vec{i}} - 1 \right)^2}_{\equiv \hat{\mathcal{H}}_U}. \tag{71}$$

This from is included in Eq. 4 such 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:

$$\hat{\mathcal{H}} = \hat{\mathcal{H}}_t + J \sum_{\vec{i}} \left(\hat{\vec{S}}^{c}_{\vec{i}} \cdot \hat{\vec{S}}^{f}_{\vec{i}} + \hat{\eta}^{z,c}_{\vec{i}} \cdot \hat{\eta}^{z,f}_{\vec{i}} - \hat{\eta}^{x,c}_{\vec{i}} \cdot \hat{\eta}^{x,f}_{\vec{i}} - \hat{\eta}^{y,c}_{\vec{i}} \cdot \hat{\eta}^{y,f}_{\vec{i}} \right) + \hat{\mathcal{H}}_U. \tag{72}$$

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

$$\hat{\eta}^{\alpha}_{\vec{i}} = \hat{P}^{-1} \hat{S}^{\alpha}_{\vec{i}} \hat{P} \text{ with } \hat{P}^{-1} \hat{c}_{\vec{i},\uparrow} \hat{P} = (-1)^{i_x + i_y} \hat{c}^{\dagger}_{\vec{i},\uparrow} \text{ and } \hat{P}^{-1} \hat{c}_{\vec{i},\downarrow} \hat{P} = \hat{c}_{\vec{i},\downarrow} \tag{73}$$

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,

$$\left[\hat{\mathcal{H}}, \hat{\mathcal{H}}_U \right] = 0. \tag{74}$$

Thereby, and for positive values of U , doubly occupied or empty f-sites corresponding to even parity will be suppressed by a Boltzmann factor $e^{-\beta U/2}$ is comparison to odd parity ones. Choosing βU adequately will essentially allow 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 reduces to the Kondo lattice model.

5.1.2 SU(N) Hubbard-Heisenberg models

SU(2N) Hubbard-Heisenberg [20, 21] models can be written as:

$$\hat{\mathcal{H}} = \underbrace{-t\sum_{\langle\vec{i},\vec{j}\rangle}\left(\vec{\hat{c}}^{\dagger}_{\vec{i}}\vec{\hat{c}}_{\vec{j}} + \text{H.c.}\right)}_{\equiv\hat{\mathcal{H}}_t} \underbrace{-\frac{J}{2N}\sum_{\langle\vec{i},\vec{j}\rangle}\left(\hat{D}^{\dagger}_{\vec{i},\vec{j}}\hat{D}_{\vec{i},\vec{j}} + \hat{D}_{\vec{i},\vec{j}}\hat{D}^{\dagger}_{\vec{i},\vec{j}}\right)}_{\equiv\hat{\mathcal{H}}_J} + \underbrace{\frac{U}{N}\sum_{\vec{i}}\left(\vec{\hat{c}}^{\dagger}_{\vec{i}}\vec{\hat{c}}_{\vec{i}} - \frac{N}{2}\right)^2}_{\equiv\hat{\mathcal{H}}_U} \qquad (75)$$

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

$$\hat{\mathcal{H}}_J = -\frac{J}{4N}\sum_{\langle\vec{i},\vec{j}\rangle}\left(\hat{D}^{\dagger}_{\langle\vec{i},\vec{j}\rangle} + \hat{D}_{\langle\vec{i},\vec{j}\rangle}\right)^2 - \left(\hat{D}^{\dagger}_{\langle\vec{i},\vec{j}\rangle} - \hat{D}_{\langle\vec{i},\vec{j}\rangle}\right)^2, \qquad (76)$$

so to manifestly bring it into the form of Eq. 4. It is amusing to note that setting the hopping $t = 0$, charge fluctuations will be suppressed by the Boltzmann factor $e^{\beta U/N\left(\vec{\hat{c}}^{\dagger}_{\vec{i}}\vec{\hat{c}}_{\vec{i}} - \frac{N}{2}\right)^2}$ since in this case $\left[\hat{\mathcal{H}}, \hat{\mathcal{H}}_U\right] = 0$. 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 [4] For odd values of N recent progress in our understanding of the origins of the sign problem [4] will allows us to simulate – without encountering the sign problem – a set of non-trivial Hamiltonians [22, 16].

5.1.3 Hubbard model in the canonical ensemble

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

$$\hat{\mathcal{H}} = \hat{\mathcal{H}_{tU}} + \underbrace{\lambda\left(\hat{N} - N\right)^2}_{\equiv\hat{H}_\lambda} \qquad (77)$$

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

$$S(\vec{q} = 0) = \sum_{\vec{r}}\left[\langle\hat{n}_{\vec{r}}\hat{n}_{\vec{0}}\rangle - \langle\hat{n}_{\vec{r}}\rangle\langle\hat{n}_{\vec{0}}\rangle\right] \qquad (78)$$

are suppressed and the grand-canonical simulation maps onto a canonical one. To implement this in the code 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:

$$e^{-\beta\hat{H}} = \prod_{\tau=1}^{L_{\text{Trotter}}}\left[e^{-\Delta\tau\hat{H}_t}e^{-\Delta\tau\hat{H}_U}\underbrace{e^{-\frac{\Delta\tau}{n_\lambda}\hat{H}_\lambda}\cdots e^{-\frac{\Delta\tau}{n_\lambda}\hat{H}_\lambda}}_{n_\lambda\text{-times}}\right]. \qquad (79)$$

Thereby per time slice we need n_λ fields 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`

5.2 Performance

Next to the entire computational time is spent in BLAS routines such that the performance of the code will depend on the implementation of this library. We have found that the code performs well, and that an efficient OpenMP version can be obtained merely by loading the corresponding BLAS and LAPACK routines.

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

6 Conclusions & 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 [23, 24, 25, 26]. Generalizations to include global moves are equally desirable. This is a prerequisite to play with recent ideas of self-learning algorithms [27] 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 [28, 29, 30] – 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, J. Schwab, T. Sato, Z. Wang and M. Weber, for constant support during the development of this project. 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 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 during the Porting and Tuning Workshop 2016 offered by the Forschungszentrum Jülich. Calculations to extensively test this package were carried out on SuperMUC at the Leibniz Supercomputing Centre and on JURECA [31] at the Jülich Supercomputing Centre (JSC). We thank those institutions for generous computer allocations.

References

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

[2] 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.

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

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

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

[6] C. J. Geyer, in *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, edited by N. Y. A. S. Association (PUBLISHER, ADDRESS, 1991), pp. 156–163.

[7] Z. Bai, C. Lee, R.-C. Li, and S. Xu, Linear Algebra and its Applications **435**, 659–673 (2011).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

License

COPYRIGHT