# The *ALF* (*Al*gorithms for *L*attice *F*ermions) project release 1.0

**Documentation for the auxiliary field quantum Monte Carlo code.**

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

December 13, 2016

## Contents

# 1 Introduction

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 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 based insights that allow one to find sign free formulations of the problem thus allowing solutions in polynomial time [3, 4]. The aim of this project is to introduce a general formulation of the finite temperature auxiliary field method so as to quickly be able to play with different model Hamiltonians at minimal programming cost. The reader is expected to be 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 will briefly list the most important equations of the auxiliary field QMC 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.

The first and most important part is to define a general Hamiltonian that can accommodate a large class of models (see Sec. 2). 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 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 thereby 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. To use the code, one will require a minimal understanding of the algorithm. In Section 2, we very briefly show how to go through the steps required to formulated the many body imaginary time propagation in terms of a sum over HS and Ising fields of one body imaginary time propagator. The user will have to provide this one body imaginary time propagator for a given configuration of HS and Ising fields.

Section 3 is devoted to the data structures which need to implement the model. This 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 equal time, time displaced and scalar observables.

After a description of the file structure in Sec 4, we will give explicit examples on how to use the code for the Hubbard model on square and honeycomb lattices for different choices of the Hubbard Stratonovich transformation (See Secs. 5, 6 and 7) as well as the Hubbard model on a square lattice coupled to a transverse Ising field (see Sec. 8 ).

The Monte Carlo run and the data analysis are separate: the QMC run dumps the results of *bins* sequentially into files which are then analyzed by analysis programs In Sec. 10, we provide a brief description of the analysis programs for our three observable types. The analysis program allow for omitting a given number of initial bins so as to allow for warmup and to rebin so as to a posteriori take into account long autocorrelation times. Finally, Sec. 11 will provide all details required to compile and run the code.

# 2 Definition of the model Hamiltonian

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

$$\hat{\mathcal{H}} = \hat{\mathcal{H}}_T + \hat{\mathcal{H}}_V + \hat{\mathcal{H}}_I + \hat{\mathcal{H}}_{0,I} \text{ , where} \tag{1}$$

$$\hat{\mathcal{H}}_T = \sum_{k=1}^{M_T} \sum_{s=1}^{N_{\mathrm{fl}}} \sum_{\sigma=1}^{N_{\mathrm{col}}} \sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}^{\dagger}_{x\sigma s} T^{(ks)}_{xy} \hat{c}_{y\sigma s} \equiv \sum_{k=1}^{M_T} \hat{T}^{(k)} \tag{2}$$

$$\hat{\mathcal{H}}_V = -\sum_{k=1}^{M_V} U_k \left\{ \sum_{s=1}^{N_{\mathrm{fl}}} \sum_{\sigma=1}^{N_{\mathrm{col}}} \left[ \left( \sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}^{\dagger}_{x\sigma s} V^{(ks)}_{xy} \hat{c}_{y\sigma s} \right) - \alpha_{ks} \right] \right\}^2 \equiv -\sum_{k=1}^{M_V} U_k \left( \hat{V}(k) \right)^2 \tag{3}$$

$$\hat{\mathcal{H}}_I = \sum_{k=1}^{M_I} \hat{Z}_k \left\{ \sum_{s=1}^{N_{\mathrm{fl}}} \sum_{\sigma=1}^{N_{\mathrm{col}}} \left[ \sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}^{\dagger}_{x\sigma s} I^{(ks)}_{xy} \hat{c}_{y\sigma s} \right] \right\} \equiv \sum_{k=1}^{M_I} \hat{Z}_k \hat{I}^{(k)} \ . \tag{4}$$

The indices have the following meaning:

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

  $N_{\mathrm{dim}}$ is the total number of spacial vertices: $N_{\mathrm{dim}} = N_{\text{unit cell}} N_{\text{orbital}}$, where $N_{\text{unit cell}}$ is the number of unit cells of the underlying Bravais lattice and $N_{\text{orbital}}$ 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 is only relevant when the Monte Carlo update probability is computed in the code (see Sec. 8 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 $\sigma$. The color index $\sigma$ only appears in the second quantized operators such that the Hamiltonian is manifestly $\mathrm{SU}(N_{\mathrm{col}})$ symmetric. We also require the matrices $\boldsymbol{T}^{(ks)}$, $\boldsymbol{V}^{(ks)}$ and $\boldsymbol{I}^{(ks)}$ to be Hermitian.

## 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 $\eta$ and $\gamma$ 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:

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

  Here, the Ising spins take the values $s_{i,\tau} = \pm1$ and the Hubbard-Stratonovich fields take the values $l_{j,\tau} = \pm2, \pm1$.

### 2.1.1 The partition function

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

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

In the above, the trace Tr runs over the Ising spins as well as over the fermionic degrees of freedom, and $\mathrm{Tr_F}$ 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:

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

All in all, the partition function is given by:

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

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

### 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 $\tau$ and $\tau'$ 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. 3.3) 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 such, 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

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

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

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

The average sign is

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

[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.

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

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

# 3 Implementation of the model

In general, the module `Hamiltonian` defines the model Hamiltonian, the lattice under consideration and the desired observables (Table 1). We have collected a number of example Hamiltonians, lattices and observables in the file `Hamiltonian_Examples.f90`. They are described in the Sec. 5 - 8. 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 be treated as as a black box.

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

| 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, 3.1.1 |
| `Ham_V` | Sets the interaction terms $\hat{\mathcal{H}}_V$ and $\hat{\mathcal{H}}_I$ by calling `Op_make` and `Op_set`. | 3.1, 3.1.1 |
| `Ham_Latt` | Sets the lattice by calling `Make_Lattice`. | 3.2 |
| `S0` | A function which returns an update ratio for the Ising term $\hat{\mathcal{H}}_{I,0}$. | 8.2 |
| `Alloc_obs` | Asigns memory storage to the observables | |
| `Obser` | Computes the scalar observables and equal-time correlation functions. | 3.3 |
| `ObserT` | Computes time-displaced correlation functions. | 3.3 |
| `Init_obs` | Initializes the observables to zero. | |
| `Pr_obs` | Writes the observables to the disk by calling `Print_bin`. | |

Table 1: 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 The `Operator` type

The fundamental data structure in the code is the derived data type `Operator`. 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 an representative of each of the above three matrices . Let us denote with $\{z_1, \cdots, z_N\}$ a subset of $N$ indices, for which

$$X_{x,y} = \begin{cases} X_{x,y} & \text{if } x, y \in \{z_1, \cdots z_N\} \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

We define the $N \times N_{\text{dim}}$ matrices $\mathbf{P}$ as

$$P_{i,x} = \delta_{z_i,x} \, , \tag{21}$$

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

$$\boldsymbol{X} = \boldsymbol{P}^T \boldsymbol{O} \boldsymbol{P} \, , \tag{22}$$

such that:

$$X_{x,y} = \sum_{i,j}^{N} P_{i,x} O_{i,j} P_{j,y} = \sum_{i,j}^{N} \delta_{z_i,x} O_{ij} \delta_{z_j,y} \, . \tag{23}$$

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

$$P_i = z_i. \tag{24}$$

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

$$e^{\boldsymbol{X}} = e^{\boldsymbol{P}^T \boldsymbol{O} \boldsymbol{P}} = \sum_{n=0}^{\infty} \frac{\left(\boldsymbol{P}^T \boldsymbol{O} \boldsymbol{P}\right)^n}{n!} = \boldsymbol{P}^T e^{\boldsymbol{O}} \boldsymbol{P} \tag{25}$$

since

$$\boldsymbol{P}\boldsymbol{P}^T = 1_{N\times N}. \tag{26}$$

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.

### 3.1.1 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 | projection matrix $\mathbf{P}$ encoded as a vector of dimension $N$. |
| `Op_X%g` | Complex | coupling strength $g$ |
| `Op_X%alpha` | Complex | constant $\alpha$ |
| `Op_X%type` | Integer | parameter to set the type of HS transformation (1 = Ising, 2 = Discrete HS, for perfect square) |
| `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 2: 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 order to specify the Hamiltonian (1), we will need several arrays of the object `Operator`. Its member variables are listed in Table 2. Since the implementation exploits the $SU(N_{\mathrm{col}})$ invariance of the Hamiltonian, we have dropped the color index $\sigma$ in the following.

- Hopping Hamiltonian (2): In this case $\boldsymbol{X} = \boldsymbol{T}^{(k,s)}$. The corresponding array of structure variables `Op_T` is `Op_T(`$M_T$`,`$N_{fl}$`)` . Precisely, a single variable `Op_T` describes the operator matrix:

$$\left( \sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}^{\dagger}_x T^{(ks)}_{xy} \hat{c}_y \right) , \tag{27}$$

  where $k = [1, M_T]$ and $s = [1, N_{\mathrm{fl}}]$. We have $g = -\Delta\tau$, $\alpha = 0$, and the type variable `Op_T%type` is irrelevant.

- Interaction Hamiltonian (3): If the interaction is of perfect-square type, we set $\boldsymbol{X} = \boldsymbol{V}^{(k,s)}$ and define the corresponding structure variables `Op_V` using the array `Op_V(M_V,N_fl)`. A single variable `Op_V` describes the operator matrix:

$$\left[ \left( \sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}^{\dagger}_x V^{(ks)}_{x,y} \hat{c}_y \right) - \alpha_{ks} \right] , \tag{28}$$

  where $k = [1, M_V]$ and $s = [1, N_{\mathrm{fl}}]$. For the perfect-square interaction, $\alpha = \alpha_{ks}$ and $g = \sqrt{\Delta\tau U_k}$. The discrete Hubbard-Stratonovich decomposition is selected by setting the type variable to `Op_V%type` $= 2$.

- Ising interaction Hamiltonian (4): In this case, $\boldsymbol{X} = \boldsymbol{I}^{(k,s)}$ and we define the array `Op_V(M_I,N_fl)`. A single variable `Op_V` then describes the operator matrix:

$$\left( \sum_{x,y}^{N_{\mathrm{dim}}} \hat{c}^{\dagger}_{x} I^{(ks)}_{xy} \hat{c}_{y} \right) , \tag{29}$$

  where $k = [1, M_I]$ and $s = [1, N_{\mathrm{fl}}]$. The Ising interaction is specified by setting the type variable `Op_V%type=1`, $\alpha = 0$ and $g = -\Delta\tau$.

- In case of a full interaction [perfect-square term (3) and Ising term (4)], we define the corresponding doubled array `Op_V(`$\mathtt{M}_V$`+`$\mathtt{M}_I$`,`$\mathtt{N}_{fl}$`)` and set the variables separately for both ranges of the array according to the above.

## 3.2 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:

$$\hat{c}_{\vec{i}+\vec{L}_1} = \hat{c}_{\vec{i}+\vec{L}_2} = \hat{c}_{\vec{i}} \tag{30}$$

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:

$$\vec{a}_i \cdot \vec{g}_i = 2\pi \delta_{i,j}, \tag{31}$$

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:

$$\begin{bmatrix} \vec{L}_1 \cdot \vec{g}_1 & \vec{L}_1 \cdot \vec{g}_2 \\ \vec{L}_2 \cdot \vec{g}_1 & \vec{L}_2 \cdot \vec{g}_2 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = 2\pi \begin{bmatrix} n \\ m \end{bmatrix} \tag{32}$$

such that

$$\begin{aligned} \vec{k} = n\vec{b}_1 + m\vec{b}_2 \text{ with } \qquad \vec{b}_1 &= \frac{2\pi}{(\vec{L}_1 \cdot \vec{g}_1)(\vec{L}_2 \cdot \vec{g}_2) - (\vec{L}_1 \cdot \vec{g}_2)(\vec{L}_2 \cdot \vec{g}_1)} \left[ (\vec{L}_2 \cdot \vec{g}_2)\vec{g}_1 - (\vec{L}_2 \cdot \vec{g}_1)\vec{g}_2 \right] \text{ and} \\ \vec{b}_2 &= \frac{2\pi}{(\vec{L}_1 \cdot \vec{g}_1)(\vec{L}_2 \cdot \vec{g}_2) - (\vec{L}_1 \cdot \vec{g}_2)(\vec{L}_2 \cdot \vec{g}_1)} \left[ (\vec{L}_1 \cdot \vec{g}_1)\vec{g}_2 - (\vec{L}_1 \cdot \vec{g}_2)\vec{g}_1 \right] \end{aligned} \tag{33}$$

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

$$S(\vec{k}, :, :, :) = \frac{1}{N_{unit\,cell}} \sum_{\vec{i},\vec{j}} e^{-i\vec{k}\cdot(\vec{i}-\vec{j})} S(\vec{i}-\vec{j}, :, :, :) \tag{34}$$

and `Fourier_K_to_R` the inverse Fourier transform

$$S(\vec{r}, :, :, :) = \frac{1}{N_{unit\,cell}} \sum_{\vec{k}\in BZ} e^{i\vec{k}\cdot\vec{r}} S(\vec{k}, :, :, :). \tag{35}$$

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

| 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. 31) |
| Latt%b1_p, Latt%b1_p | Real | k-quantization (See Eq. 33) |
| 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 3: 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 will be generated when calling: `Call Make_Lattice( L1, L2, a1, a2, Latt )`.

## 3.3 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 a 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 I/O. We also obtain improved estimators by taking measurements in the imaginary-time interval `[LOBS_ST,LOBS_EN]` (see the parameter file in Sec. 4.1) 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 will define bins, each bins having a given amount of sweeps. Within a sweep we run sequentially trough the HS and Ising fields from time slice 1 to $L_{\text{Trotter}}$ and back. The results of each bin is written in 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 measurement of an observable has to be multiplied by the factors `ZS` and `ZP`:

$$\texttt{ZS} = \text{sign}(C) \tag{36}$$

$$\texttt{ZP} = \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]}, \tag{37}$$

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

$$\texttt{phase} = \frac{e^{-S(C)}}{\left|e^{-S(C)}\right|}, \tag{38}$$

which is provided by the main program.

Note that each observable structure also includes the average sign [Eq. (18)].

### 3.3.1 Scalar observables

This data type is described in Table 4 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)] | $\mathrm{sign}(C)$ |
| `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 4: 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.

### 3.3.2 Equal time and time-displaced correlation functions

| Variable | Type | Description | Contribution of configuration $C$ |
|---|---|---|---|
| `Obs%N` | Int. | Number of measurements | |
| `Obs%Ave_sign` | Real | Cumulated sign [Eq. (18)] | $\mathrm{sign}(C)$ |
| `Obs%Obs_latt` $(\vec{i}-\vec{j},\tau,\alpha,\beta)$ | Compl. | Cumul. correl. fct. [Eq. (17)] | $\langle\langle \hat{O}_{\vec{i},\alpha}(\tau)\hat{O}_{\vec{j},\beta}\rangle\rangle_C \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]}\mathrm{sign}(C)$ |
| `Obs%Obs_latt0`$(\alpha)$ | 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 5: Components of the `Obser_latt` type. The table lists the data included in a variable `Obs` of type `Obser_latt`

This data type (see Table 5) is useful so as to deal with imaginary time displaced as well as equal time correlation functions of the form:

$$S_{\alpha,\beta}(\vec{k},\tau) = \frac{1}{N_{\text{unit cell}}}\sum_{\vec{i},\vec{j}} e^{-\vec{k}\cdot(\vec{i}-\vec{j})}\left(\langle \hat{O}_{\vec{i},\alpha}(\tau)\hat{O}_{\vec{j},\beta}\rangle - \langle \hat{O}_{\vec{i},\alpha}\rangle\langle \hat{O}_{\vec{i},\beta}\rangle\right). \tag{39}$$

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

$$\begin{aligned} S^{(\mathrm{corr})}_{\alpha,\beta}(\vec{k},\tau) &= \frac{1}{N_{\text{unit cell}}}\sum_{\vec{i},\vec{j}} e^{-i\vec{k}\cdot(\vec{i}-\vec{j})}\langle \hat{O}_{\vec{i},\alpha}(\tau)\hat{O}_{\vec{j},\beta}\rangle\,, & (40)\\ S^{(\mathrm{back})}_{\alpha,\beta}(\vec{k}) &= \frac{1}{N_{\text{unit cell}}}\sum_{\vec{i},\vec{j}} e^{-i\vec{k}\cdot(\vec{i}-\vec{j})}\langle \hat{O}_{\vec{i},\alpha}(\tau)\rangle\langle \hat{O}_{\vec{j},\beta}\rangle \\ &= N_{\text{unit cell}}\,\langle \hat{O}_{\alpha}\rangle\langle \hat{O}_{\beta}\rangle\,\delta(\vec{k})\,, & (41)\end{aligned}$$

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

$$\langle \hat{O}_\alpha\rangle \equiv \frac{1}{N_{\text{unit cell}}}\sum_{\vec{i}}\langle \hat{O}_{\vec{i},\alpha}\rangle\,. \tag{42}$$

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 $\langle\langle \hat{O}_{\vec{i},\alpha}(\tau)\hat{O}_{\vec{j},\beta}\rangle\rangle_C \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]}$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$(\alpha)$. 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 , that is the second dimension of the array Obs%Obs_latt$(\vec{i}-\vec{j},\tau,\alpha,\beta)$ is equal 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.

# 4 File structure

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

Table 6: Overview of the directories.

The code package consists of the program directories Prog/, Libraries/ and Analysis/. The sample simulations corresponding to the walkthroughs of Sec. 5 - 8 are included in Examples/. The package content is summarized in Table 6.

## 4.1 Input files

| File | Description |
|---|---|
| parameters | Sets the parameters for lattice, model, QMC process, and the error analysis. |
| seeds | List of integer numbers to initialize the random number generator and to start a simulation from scratch. |
| confin_<threadnumber> | Input files for the HS and Ising configuration, used to continue a simulation. |

Table 7: Overview of the input files in Start/ required for a simulation.

The input files are listed in Table 7. The parameter file Start/parameters has the following form, using as an example the $SU(2)$-symmetric Hubbard model on a square lattice (see Sec. 5 for a detailed walkthrough):

```
 ================================================================================
!  Variables for the Hubb program
!--------------------------------------------------------------------------------
&VAR_lattice
L1 = 4                     ! Length in direction a_1
L2 = 4                     ! Length in direction a_2
Lattice_type = "Square"    ! a_1 = (1,0),  a_2=(0,1),  Norb=1, N_coord=2
!Lattice_type ="Honeycomb"! a_1 = (1,0),  a_2 =(1/2,sqrt(3)/2), Norb=2, N_coord=3
Model = "Hubbard_SU2"      ! Sets Nf=1, N_sun=2. HS field couples to the density
!Model = "Hubbard_Mz"      ! Sets Nf=2, N_sun=1. HS field couples to the
                           ! z-component of magnetization.
!Model="Hubbard_SU2_Ising"! Sets Nf_1, N_sun=2 and runs only for the square lattice
                           ! Hubbard model  coupled to transverse Ising field
```

```
/
&VAR_Hubbard             ! Variables for the Hubbard model
ham_T    = 1.D0          ! Hopping parameter
ham_chem= 0.D0           ! chemical potential
ham_U    = 4.D0          ! Hubbard interaction
Beta     = 5.D0          ! inverse temperature
dtau     = 0.1D0         ! Thereby Ltrot=Beta/dtau
/

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

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

## 4.2 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, CPU 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 8: Overview of the standard output files. See Sec. 3.3 for the definitions of observables and correlation functions.

The output of the measured data is organized in bins. One bin corresponds to the geometric 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 run a parallelized version of the code, the average also extends over the number of MPI threads. The standard output files are listed in Table 8.

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 phase>
```

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. 10). 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. (42)] contributing to the background part [Eq. (41)] of the correlation function, and the correlated part [Eq. (40)] of the correlation function. For imaginary-time displaced correlation functions, the formatting of the block follows this scheme:

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

The same block structure is used for equal-time correlation functions, except for the entries `<N_time_slices>` and `<dtau>` which are not present in the latter. Using this structure for the bins as input, the full correlation function $S_{\alpha,\beta}(\vec{k},\tau)$ [Eq. (39)] is then calculated by calling the error analyis routine (see Sec. 10)

### 4.2.1 The `info` file and stabilization

The finite temperature auxiliary field QMC algorithm is known to be numerically unstable. The origin the numerical instabilities arise 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. [5] for the finite temperature algorithm and in Refs.[6, 7] for the zero 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` simulating the $4 \times 4$ 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 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 coupling 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.

Since numerical stabilization is delicate we have include in the package the set of Lapack version 3.6.1 routines which we use for purposes of numerical stabilization. These routines will be used if the compilation flag QRREF is set.

## 4.3 Scripts

| Script | Description | Section |
|---|---|---|
| `set_env.sh` | Sets the environment variables for the compiler and the libraries. | 11 |
| `Start/out_to_in.sh` | Copies the output configurations of HS and Ising spins to the respective input files. | 11 |
| `Start/analysis.sh` | Starts the error analysis. | 10 |

Table 9: Overview of the bash script files.

# 5 Walkthrough: 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

$$\mathcal{H} = \sum_{\sigma=1}^{2} \sum_{x,y=1}^{N_{\text{unit cell}}} c^{\dagger}_{x\sigma} T_{x,y} c_{y\sigma} + \frac{U}{2} \sum_{x} \left[ \sum_{\sigma=1}^{2} \left( c^{\dagger}_{x\sigma} c_{x\sigma} - 1/2 \right) \right]^{2} . \tag{43}$$

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

## 5.1 Setting the Hamiltonian: `Ham_set`

The main program will call the subroutine `Ham_set` in the module `Hamiltonian_Hub.f90`. This 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. This routine 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. 4.1.

### 5.1.1 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 $N_{\text{dim}} \equiv N_{\text{unit cell}} \cdot \texttt{NORB} = \texttt{Latt\%N}$.

### 5.1.2 The hopping term: Call Ham_hop

The hopping matrix is implemented as follows. We allocate an array of dimension $1 \times 1$ 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 $\boldsymbol{P} = \mathbb{1}$ 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. Note that although a checkerboard decomposition is not used here, it can be implemented by considering a larger number of sparse hopping matrices

### 5.1.3 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
Enddo
```

so as to completely define the interaction term.

## 5.2 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

The user will have to implement the observables he/she wants to compute. Here we will describe how to proceed.

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

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

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

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

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

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

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

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

The equal time green function,

$$\texttt{GR(x,y,}\sigma\texttt{)} = \langle c_{x,\sigma} c^{\dagger}_{y,\sigma} \rangle, \tag{44}$$

the phase factor `phase` [Eq. (38)] 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` $=$ $\text{sign}(C)$ [see Eq. (16)], `ZP` $= \frac{e^{-S(C)}}{\Re\left[e^{-S(C)}\right]}$ [see Eq. (38)] 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:

$$\langle n_{\vec{i},\alpha} n_{\vec{j},\beta} \rangle - \langle n_{\vec{i},\alpha} \rangle \langle n_{\vec{j},\beta} \rangle \tag{45}$$

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

$$\texttt{GRC(x,y,s)} = \delta_{x,y} - \texttt{GR(y,x,s)} \tag{46}$$

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)  * NSUN * NSUN       + &
                     &          GRC(I1,J1,1)  * GR (I1,J1,1   * NSUN              )* ZP*ZS
   Enddo
   Obs_eq(4)%Obs_Latt0(no_I) =  Obs_eq(4)%Obs_Latt0(no_I) +   GRC(I1,I1,1) * NSUN * 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.

### 5.2.3 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:

$$\begin{aligned}
\texttt{GT0(x,y,s)} &= \langle\langle \hat{c}_{x,s}(Nt\Delta\tau) \hat{c}^{\dagger}_{y,s}(0) \rangle\rangle = \langle\langle \mathcal{T} \hat{c}_{x,s}(Nt\Delta\tau) \hat{c}^{\dagger}_{y,s}(0) \rangle\rangle \\
\texttt{G0T(x,y,s)} &= -\langle\langle \hat{c}^{\dagger}_{y,s}(Nt\Delta\tau) \hat{c}_{x,s}(0) \rangle\rangle = \langle\langle \mathcal{T} \hat{c}_{x,s}(0) \hat{c}^{\dagger}_{y,s}(Nt\Delta\tau) \rangle\rangle \\
\texttt{G00(x,y,s)} &= \langle\langle \hat{c}_{x,s}(0) \hat{c}^{\dagger}_{y,s}(0) \rangle\rangle \\
\texttt{GTT(x,y,s)} &= \langle\langle \hat{c}_{x,s}(Nt\Delta\tau) \hat{c}^{\dagger}_{y,s}(Nt\Delta\tau) \rangle\rangle
\end{aligned} \tag{47}$$

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:

$$4\langle\langle \hat{S}^z_{\vec{i}}(\tau)\hat{S}^z_{\vec{j}}(0)\rangle\rangle = -2\,\texttt{GOT(J,I,1)}\ \texttt{GTO(I,J1)} \qquad (48)$$

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.

# 6 Walkthrough: the $M_z$-Hubbard model on a square lattice

The Hubbard Hamiltonian can equally be written as:

$$\mathcal{H} = \sum_{\sigma=1}^{2} \sum_{x,y=1}^{N_{unit\ cells}} c^\dagger_{x\sigma} T_{x,y} c_{y\sigma} - \frac{U}{2}\sum_{x}\left[c^\dagger_{x,\uparrow}c_{x\uparrow} - c^\dagger_{x,\downarrow}c_{x\downarrow}\right]^2 . \qquad (49)$$

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. 4.1 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 favor 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, 8, 9]. A sample run for this model can be found in `Examples/Hubbard_Mz_Square/`.

## 6.1 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.

### 6.2 The measurements: `Call Obser`, `Call ObserT`

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

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

Here, `GRC` is defined in Eq. 46. 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.

## 7 Walkthrough: 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. 4.1) `Lattice_type = "Honeycomb"`. A sample run for this model can be found in `Examples/Hubbard_SU2_Honeycomb/`.

### 7.1 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: $N_{\mathrm{dim}}$=`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.

### 7.2 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.

### 7.3 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.

## 8 Walkthrough: the $SU(2)$-Hubbard model on a square lattice coupled to a transverse Ising field

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

$$\begin{aligned}\mathcal{H} = \quad & \sum_{\sigma=1}^{2}\sum_{x,y} c^{\dagger}_{x\sigma} T_{x,y} c_{y\sigma} + \frac{U}{2}\sum_{x}\left[\sum_{\sigma=1}^{2}\left(c^{\dagger}_{x\sigma}c_{x\sigma} - 1/2\right)\right]^2 + \xi \sum_{\sigma,\langle x,y\rangle} \hat{Z}_{\langle x,y\rangle}\left(c^{\dagger}_{x\sigma}c_{y\sigma} + h.c.\right) \\ & -h\sum_{\langle x,y\rangle}\hat{X}_{\langle x,y\rangle} - J\sum_{\langle\langle x,y\rangle\langle x',y'\rangle\rangle}\hat{Z}_{\langle x,y\rangle}\hat{Z}_{\langle x',y'\rangle} \end{aligned} \tag{50}$$

We can make contact with the general form of the Hamiltonian by setting: $N_{\mathrm{fl}} = 1$, $N_{\mathrm{col}} \equiv$ `N_SUN` $= 2$, $M_T = 1$, $T^{(ks)}_{xy} = T_{x,y}$, $M_V = N_{\text{unit cell}} \equiv N_{\mathrm{dim}}$, $U_k = -\frac{U}{2}$, $V^{(ks)}_{xy} = \delta_{x,y}\delta_{x,k}$, $\alpha_{ks} = \frac{1}{2}$ and $M_I = 2N_{\text{unit cell}}$. The last two terms of the above Hamiltonian describes a transverse Ising field model on the bonds of the square lattice. This type of Hamiltonian has recently been extensively discussed [10, 11, 12]. Here we adopt the notation of Ref. [12]. 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/`.

### 8.1 The interaction term: `Call Ham_V`

The dimension of `Op_V` is now $(M_I + M_V) \times N_{\mathrm{fl}} = (3 * N_{\mathrm{dim}}) \times 1$. 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 nc  = 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
```

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

As mentioned above, a configuration is given by

$$C = \{s_{i,\tau}, l_{j,\tau} \text{ with } i = 1\cdots M_I, j = 1\cdots M_V, \tau = 1, L_{Trotter}\} \tag{51}$$

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 `Ndim` fields corresponds to the Hubbard interaction and the next `2*Ndim` ones to the Ising 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

$$\frac{e^{-S_{0,I}\left(s_{1,\tau},\cdots,-s_{m,\tau},\cdots s_{M_I,\tau}\right)}}{e^{-S_{0,I}\left(s_{1,\tau},\cdots,s_{m,\tau},\cdots s_{M_I,\tau}\right)}} \tag{52}$$

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 - Ndim`

# 9 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.

## 9.1 The Kondo lattice

Simulating the Kondo lattice within the ALF-project requires rewriting of the model along the lines of Refs. [13, 14, 15]. Adopting the notation of these articles, the Hamiltonian that one will simulate reads:

$$\hat{H} = \underbrace{-t \sum_{\langle \vec{i},\vec{j}\rangle,\sigma} \left( \hat{c}^{\dagger}_{\vec{i},\sigma} \hat{c}_{\vec{j},\sigma} + \text{H.c.} \right)}_{\equiv \hat{H}_t} - \frac{J}{4} \sum_{\vec{i}} \left( \sum_{\sigma} \hat{c}^{\dagger}_{\vec{i},\sigma} \hat{f}_{\vec{i},\sigma} + \hat{f}^{\dagger}_{\vec{i},\sigma} \hat{c}_{\vec{i},\sigma} \right)^2 + \underbrace{\frac{U}{2} \sum_{\vec{i}} \left( \hat{n}^{f}_{\vec{i}} - 1 \right)^2}_{\equiv \hat{H}_U}. \tag{53}$$

This from is included in Eq. 4 such the above Hamiltonian can be implemented in our program package. The relation to the Kondo lattice mdel follows from expanding the square of the hybridization to obtain:

$$\hat{H} = \hat{H}_t + J \sum_{\vec{i}} \left( \hat{\vec{S}}^{c}_{\vec{i}} \cdot \hat{\vec{S}}^{f}_{\vec{i}} + \hat{\eta}^{z,c}_{\vec{i}} \cdot \hat{\eta}^{z,f}_{\vec{i}} - \hat{\eta}^{x,c}_{\vec{i}} \cdot \hat{\eta}^{x,f}_{\vec{i}} - \hat{\eta}^{y,c}_{\vec{i}} \cdot \hat{\eta}^{y,f}_{\vec{i}} \right) + \hat{H}_U. \tag{54}$$

where the $\eta$-operators related to the spin-operators via a particle-hole transformation in one spin sector:

$$\hat{\eta}^{\alpha}_{\vec{i}} = \hat{P}^{-1} \hat{S}^{\alpha}_{\vec{i}} \hat{P} \ \text{ with } \ \hat{P}^{-1} \hat{c}_{\vec{i},\uparrow} \hat{P} = (-1)^{i_x + i_y} \hat{c}^{\dagger}_{\vec{i},\uparrow} \ \text{ and } \ \hat{P}^{-1} \hat{c}_{\vec{i},\downarrow} \hat{P} = \hat{c}_{\vec{i},\downarrow} \tag{55}$$

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,

$$\left[ \hat{H}, \hat{H}_U \right] = 0. \tag{56}$$

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 $\beta U$ 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.

## 9.2 SU(N) Hubbard-Heisenberg models

SU(2N) Hubbard-Heisenberg [16, 17] models can be written as:

$$\hat{H} = \underbrace{-t \sum_{\langle \vec{i},\vec{j}\rangle} \left( \vec{\hat{c}}^{\dagger}_{\vec{i}} \vec{\hat{c}}_{\vec{j}} + \text{H.c.} \right)}_{\equiv \hat{H}_t} \underbrace{- \frac{J}{2N} \sum_{\langle \vec{i},\vec{j}\rangle} \left( \hat{D}^{\dagger}_{\vec{i},\vec{j}} \hat{D}_{\vec{i},\vec{j}} + \hat{D}_{\vec{i},\vec{j}} \hat{D}^{\dagger}_{\vec{i},\vec{j}} \right)}_{\equiv \hat{H}_J} + \underbrace{\frac{U}{N} \sum_{\vec{i}} \left( \vec{\hat{c}}^{\dagger}_{\vec{i}} \vec{\hat{c}}_{\vec{i}} - \frac{N}{2} \right)^2}_{\equiv \hat{H}_U} \tag{57}$$

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,

$$\hat{H}_J = -\frac{J}{4N} \sum_{\langle \vec{i},\vec{j}\rangle} \left( \hat{D}^{\dagger}_{\langle \vec{i},\vec{j}\rangle} + \hat{D}_{\langle \vec{i},\vec{j}\rangle} \right)^2 - \left( \hat{D}^{\dagger}_{\langle \vec{i},\vec{j}\rangle} - \hat{D}_{\langle \vec{i},\vec{j}\rangle} \right)^2, \tag{58}$$

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 $\left[ \hat{H}, \hat{H}_U \right] = 0$. This provides a route to use the auxiliary field QMC algorithm so as to simulate – free of the sign problem – SU(2N) Heisenberg models in the self-adjoint antisymmetric representation [2] 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 [18, 12].

| Program | Description |
|---|---|
| cov_scal.f90 | Reads in the bin files with suffix _scal and produces corresponding file with suffix _scalJ containing the result of the Jackknife resampling. |
| cov_eq.f90 | Reads in the bin files with suffix _eq and produces corresponding files will suffix _eqJR and _eqJK corresponding to correlation functions in real and Fourier space, respectively. |
| cov_tau.f90 | Reads in the bin files X_tau, and produces and produces directories X_kx_ky for all kx and ky greater or equal to zero. Here X is a place holder from Green, SpinXY, etc as specified in Alloc_obs(Ltau) (See section 5.2.1). 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 listed in Sec. 4.1. The program equally generates a directory X_R0 for the local time displaced correlation function. |

Table 10: Overview of analysis programs

# 10 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. This will be the case if bins are longer than the auto-correlation time. In the parameter file listed in Sec. 4.1, the user can specify the how many initial bins should be omitted (variable n_skip). This number should be comparable to the auto-correlation time. The re-binning variable N_rebin will merge N_rebin bins into a single one. If the autocorrelation time is smaller than the effective bin size, then the error should be independent on the bin size and thereby on the variable N_rebin. Our analysis is based on the Jackknife resampling. As listed in Table, 10 we provide three programs to account for the three observable types. The programs can be found in the directory Analysis and are executed by running the bash shell script analysis.sh In the following, we describe the formatting of the output files mentioned in Table 12.

| File | Description |
|---|---|
| parameters | Contains also variables for the error analysis: n_skip, N_rebin and N_Cov (see Sec. 4.1) |
| X_scal, Y_eq, Y_tau | Monte Carlo bins (see Table 8) |

Table 11: Standard input files for the error analysis.

| File | Description |
|---|---|
| X_scalJ | Jackknife mean and error of X, where X stands for Kin, Pot, Part, and Ener. |
| Y_eqJR and Y_eqJK | Jackknife mean and error of Y, where Y stands for Green, SpinZ, SpinXY, and Den. The suffixes R and K 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 12: Standard output files of the error analysis.

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

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

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

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

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

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

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

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

- The 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.

# 11 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.

## 11.1 Compilation

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

```
# setting QRREF has the highest priority. Setting nothing selects
# system lapack for the QR decomposition.
# -DMPI selects MPI.
PROGRAMMCONFIGURATION="-DQRREF"
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
```

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 ..
```

### 11.2 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 one of the Hubbard model described in Sec. 5 - 8, 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.

### 11.3 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. 10).

## 12 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.

## 13 Conclusions and future directions

In it's present form, the ALF-project allows to simulate a very large class of non-trivial models efficiently and at a minimal programming cost. There are many possible extensions which deserve to be considered in future releases. The Hamiltonians we presently defining are imaginary time independent. This however, can be easily generalized to time dependent Hamiltonians thus allowing, for example, to access entanglement properties of interacting fermionic systems [19, 20, 21, 22]. Generalizations to include global moves are equally desirable. This is a prerequisite to play with recent ideas of self-learning algorithms [23] so as to possibly avoid critical slowing down. At present we are restricted to discrete fields such that implementations of the long range Coulomb repulsion as introduced in [24, 25, 26] is not included in the package. Extensions to continuous fields are certainly possible, but require an efficient upgrading scheme. Finally, a ground state projective formulation is equally desirable.

## Acknowledgements

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 (KONWIRH) for financial support. FG thanks the SFB-1170 for financial support under project (Z03). FFA thanks the DFG-funded FOR1807 for partial financial support. Calculations to extensively test this package were carried out on SuperMUC at the Leibniz Supercomputing Centre and on JURECA [27] 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] S. White, D. Scalapino, R. Sugar, E. Loh, J. Gubernatis, and R. Scalettar, Phys. Rev. B **40**, 506 (1989).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## License

Use of the GQMC code requires citation of the paper ... The GQMC code is available for academic and non-commercial use under the terms of the license ... For commercial licenses, please contact the GQMC development team.

This package contains part of the lapack implementation version 3.6.1 from `http://www.netlib.org/lapack` lapack is licensed under the modified BSD license which we reproduce here:



### COPYRIGHT

Additional copyrights may follow

### HEADER

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ”AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.