The *ALF* (*A*lgorithms for *L*attice *F*ermions) project release 2.0

Documentation for the auxiliary field quantum Monte Carlo code.

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

September 1, 2019

Contents

1 Introduction

2 Auxiliary Field Quantum Monte Carlo

3 Data Structures and Input/Output

3.1 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 1: Overview of the directories.

The code package consists of the program directories `Prog/`, `Libraries/` and `Analysis/`. The example simulations corresponding to the walkthroughs of Sec. 4.1 - 4.4 are included in `Examples/`. The package content is summarized in Table 1.

3.1.1 Input files

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

Table 2: Overview of the input files in `Start/` required for a simulation.

The input files are listed in Table 2. 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):

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

```
dtau    = 0.1D0           ! Thereby Ltrot=Beta/dtau
/

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

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

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

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

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

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

```
&VAR_Max_stoch              ! Variables for  parallel tempering
N_exchange_steps     =6     ! Number of exchange moves (be more precise)
N_Tempering_frequency=10    ! The frequency in units of sweeps at which
                            ! the exchange moves will be carried
mpi_per_parameter_set=2     ! Number of mpi-processes per parameter set
```

```
Tempering_calc_det   =.true. ! Specifies whether the fermion weight has to
                             ! be taken into account while tempering. The
                             ! default is .true., can be set to .false. if
                             ! the parameters that get varied only enter
                             ! the Ising action S_0
/
```

3.1.2 Output: Observables

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

Table 3: Overview of the standard output files. See Sec. ?? for the definitions of observables and correlation functions.

The standard output files are listed in Table 3. The output of the measured data is organized in bins. One bin corresponds to the arithmetic average over a fixed number of individual measurements which depends on the chosen measurement interval `[LOBS_ST,LOBS_EN]` on the imaginary-time axis and on the number `NSweep` of Monte Carlo sweeps. If the user runs an MPI parallelized version of the code, the average also extends over the number of MPI threads. The formatting of the output for a single bin depends on the observable type, `Obs_vec` or `Obs_Latt`:

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

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

  The counter variable `N_size+1` refers to the number of measurements per line, including the phase measurement. This format is required by the error analysis routine (see Sec. 3.3). 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. (??)] contributing to the background part [Eq. (??)] of the correlation function, and the correlated part [Eq. (??)] of the correlation function. For imaginary-time displaced correlation functions, the formatting of the block follows this scheme:

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

The same block structure is used for equal time correlation functions, except for the entries `<N_time_slices>` and `<dtau>` which are not present in the latter. Using this structure for the bins as input, the full correlation function $S_{\alpha,\beta}(\vec{k},\tau)$ [Eq. (**??**)] is then calculated by calling the error analysis routine (see Sec. 3.3).

3.1.3 Output: Precision

The finite temperature auxiliary field QMC algorithm is known to be numerically unstable, as discussed in Sec. **??**. 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. [1] for the finite-temperature algorithm and in Refs. [2, 3] for the zero temperature projective algorithm. As shown in Ref. [4] scales can be omitted in the ground state algorithm – thus rendering it very stable – but have to be taken into account in the finite-temperature code. Apart from runtime information, the file `info` contains important information concerning the stability of the code. It is important to know that numerical stabilization is delicate and there is no guarantee that it will work for all models.

If the numerical stabilization turns out to be bad, one option is to reduce the value of the parameter `Nwrap` in the parameter file. For performing the stabilization of the involved matrix multiplications we rely on routines from LAPACK. Hence it is very likely that your results may change significantly if you switch the LAPACK implementation. In order to offer a simple baseline to which people can quickly switch if they want to see whether their results depend on the library used for linear algebra routines we have included parts of the LAPACK-3.6.1 reference implementation from `http://www.netlib.org/lapack/`. You can switch to the QR decomposition related routines from the LAPACK reference implementation by including the switch `-DQRREF` into the PROGRAMMCONFIGURATION string. To use these routines you need to link against a lapack library that implements at least the LAPACK-3.4.0 interface.[1]

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

Typical values for the numerical precision can be found in the examples of Sec. 4 (see Sec. 4.1.3 and 4.2.3).

3.2 Scripts

Script	Description	Section
`Start/out_to_in.sh`	Copies the output configurations of HS and Ising spins to the respective input files.	3.4
`Start/analysis.sh`	Starts the error analysis.	3.3

Table 4: Overview of the bash script files.

3.3 Analysis programs

Here we briefly discuss the analysis programs which read in bins and carry out the error analysis. (See Sec. **??** for a more detailed discussion.) Error analysis is based on the central limit theorem, which requires bins to be statistically independent, and also the existence of a well-defined variance for the observable under consideration. The former will be the case if bins are longer than the autocorrelation time. The latter has to be checked by the user. In the parameter file listed in Sec. 3.1.1, the user can specify how many initial bins should be omitted (variable `n_skip`). This number should be comparable to the autocorrelation time. The rebinning variable `N_rebin` will merge `N_rebin` bins into a single new bin. If the autocorrelation time is smaller than the effective bin size, the error should become independent of the bin size and thereby of the variable `N_rebin`. Our analysis is based on the Jackknife resampling[5]. As listed in Table 5 we provide three analysis programs to account for the three observable types. The

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

Program	Description
`cov_scal.f90`	In combination with the script `analysis.sh`, the bin files with suffix `_scal` are read in, and the corresponding files with suffix `_scalJ` are produced. They contain the result of the Jackknife rebinning analysis (see Sec. **??**).
`cov_eq.f90`	In combination with the script `analysis.sh`, the bin files with suffix `_eq` are read in, and the corresponding files with suffix `_eqJR` and `_eqJK` are produced. They correspond to correlation functions in real and Fourier space, respectively.
`cov_tau.f90`	In combination with the script `analysis.sh`, the bin files `X_tau` are read in, and the directories `X_kx_ky` are produced for all `kx` and `ky` greater or equal to zero. Here `X` is a place holder from `Green`, `SpinXY`, etc as specified in `Alloc_obs(Ltau)` (See section 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. 3.1.1). Equally, a directory `X_R0` for the local time displaced correlation function is generated.
`cov_tau_ph.f90`	At compilation time the file `cov_tau_ph.f90` is generated, and should be used to compute particle-hole imaginary time correlation functions such as Spin and Charge. Here we use the fact that these correlation functions are symmetric around $\tau = \beta/2$ so that we can define an improved estimator by averaging over τ and $\beta - \tau$.

Table 5: Overview of analysis programs that are called within the script `analysis.sh`.

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 7.

File	Description
`parameters`	Contains also variables for the error analysis: `n_skip`, `N_rebin` and `N_Cov` (see Sec. 3.1.1)
`X_scal`, `Y_eq`, `Y_tau`	Monte Carlo bins (see Table 3)

Table 6: Standard input files for the error analysis.

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

Table 7: Standard output files of the error analysis.

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

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

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

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

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

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

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

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

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

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

3.4 Running the code

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

3.4.1 Compilation

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

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

```
export F90USEFULFLAGS
FL = -c ${F90OPTFLAGS} ${PROGRAMCONFIGURATION}
export FL
DIR = ${CURDIR}
export DIR
Libs = ${DIR}/Libraries/
export Libs
LIB_BLAS_LAPACK = -llapack -lblas
export LIB_BLAS_LAPACK

all: lib ana program

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


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

In the above, the GNU Fortan compiler `gfortran` is set.[2] We provide a set of options for compilation of the QMC code. The present options are `-DMPI`, `-DQRREF`, `-DSTAB1`, and `-DSTAB2`. They can be included in the string variable `PROGRAMCONFIGURATION` by the user, as shown above. The program can be compiled and ran either in single-thread mode (default) or in multi-threading mode (define `-DMPI`) using the MPI standard for parallelization. The remaining three compiler options select a particular stabilization scheme for the matrix multiplications (see Sec. 3.1.3). To compile the libraries, the analysis routines and the QMC program at once, just execute the single command:

```
make
```

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

3.4.2 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.

[2] 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.

3.4.3 Error analysis

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

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

To perform an error analysis based on the Jackknife resampling method (Sec. **??**) of the Monte Carlo bins for all observables run the script `analysis.sh` (see Sec. 3.3). In case that the parameter `N_auto` is set to a finite value the script will also trigger the computation of autocorrelation functions (Sec. **??**).

3.5 Maximum entropy

3.5.1 General setup

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

$$g(\tau) = \int_{\omega_{start}}^{\omega_{end}} d\omega K(\tau,\omega) A(\omega). \tag{1}$$

The ALF-package includes a standard implementation of the stochastic MaxEnt as formulated in the article of K. Beach Ref. [6]. Here we will comment on the workflow. The module `/Libraries/Modules/maxent_stochf90` contains a general implementation and the wrapper `Analysis/Max_SAC.f90`.

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

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

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

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

$$A(\omega) = \sum_{i=1}^{N_\gamma} a_i \delta(\omega - \omega_i). \tag{3}$$

To produce a histogram of $A(\omega)$ we divide the frequency range in `Ndis` intervals. The Green function is read from the file `g_dat` corresponding to the output of the the `cov_tau.f90` analysis program. Below, we summarize the parameters in `param_SAC` required to run the code .

```
&VAR_Max_Stoch              ! Variables for Stochastic Maximum entropy
Ngamma                      ! # of  Dirac functions for parametrization
Om_st                       ! Frequency range lower bound
Om_en                       ! Frequency range upper bound
NDis                        ! # of boxes for histogram
Nbins                       ! # of bins for Monte Carlo
Nsweeps                     ! # of sweeps per bin
NWarm                       ! The Nwarm first bins will be ommitted
N_alpha                     ! # of tempertures
alpha_st                    ! smallest inverse temperature
R                           ! increment for inverse temperature (see above)
L_cov                       ! =1  covariance is taken into  account
                            ! =0  covariance is not taken into account
Channel                     ! T0       : Zero temperature
                            ! P        : Finite temperarure particle
                            ! PH       : Finite temperarure particle-hole
```

```
                              ! PP       : Finite temperarure particle-particle
Checkpoint                    !.true.    : dump files will be produces so
                              !            as to be able to restart the simulation
                              !.false.   : dump files will not be produces
/
```

Output files
The code produces the following output files.

- The files `Aom_n` correspond to the average spectral function at inverse temperature α_n. This corresponds to $\langle A_n(\omega)\rangle = \frac{1}{Z}\int DA(\omega)e^{-\alpha_n\chi^2(A)}A(\omega)$. The file contains three colums ω, $\langle A_n(\omega)\rangle$, $\Delta\langle A_n(\omega)\rangle$.
- The files `Aom_ps_n` contain the average image over the inverse temperatures α_n to α_{N_γ} see Ref. [6] for more details. The first three columns have the same meaning as for the files `Aom_n`
- The file `Green` contains the Green function. The three columns correspond to ω, $\mathrm{Re}G(\omega)$, $\mathrm{Im}G(\omega)$. This is obtained from the spectral function through:
$$G(\omega) = -\frac{1}{\pi}\int d\Omega \frac{A(\Omega)}{\omega - \Omega + i\delta} \tag{4}$$
where $\delta = \Delta\omega$ with $\Delta\omega = (\omega_{end} - \omega_{start})/\mathrm{Ndis}$ and the image corresponds to that of the file `Aom_ps_m` with $m = N_\alpha - 10$.
- One of the most important files is the file `energies`. It contains there columns: $\alpha_n, \langle\chi^2\rangle, \Delta\langle\chi^2\rangle$.
- `best_fit` gives the values of a_i and ω_i (recall that $A(\omega) = \sum_{i=1}^{N_\gamma} a_i\delta(\omega - \omega_i)$.) corresponding to the last configuration of the lowest temperature run.
- The File `data_out` is a crosscheck. It plots $\tau, g(\tau), \Delta g(\tau), \int d\omega K(\tau,\omega)A(\omega)$ where the image corresponds to the best fit (i.e. the lowest temperature). This file will give you a feeling on how good the fit actually is.
- There are two `dump` files which are generated. Since the MaxEnt is a Monte Carlo code, one would like to be able to continue a simulation to improve. The data in the dump files will allow you to pursue the simulation without loosing the first run(s). These files are only generated if the variable `checkpoint` is set to true.

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

3.5.2 Single particle quantities

For the single-particle Green function,

$$\langle\hat{c}_k(\tau)\hat{c}_k^\dagger(0)\rangle = \int d\omega K_p(\tau,\omega)A_p(k,\omega) \tag{5}$$

with

$$K_p(\tau,\omega) = \frac{1}{\pi}\frac{e^{-\tau\omega}}{1+e^{-\beta\omega}} \tag{6}$$

and

$$A_p(k,\omega) = \frac{\pi}{Z}\sum_{n,m} e^{-\beta E_n}\left(1+e^{-\beta\omega}\right)|\langle n|c_n|m\rangle|^2\delta\left(E_m - E_n - \omega\right) \tag{7}$$

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

$$G^{\mathrm{ret}}(k,\omega) = -i\int dt\Theta(t)e^{i\omega t}\langle\left\{\hat{c}_k(t),\hat{c}_k^\dagger(0)\right\}\rangle \tag{8}$$

Finally the sum rule reads:

$$\int d\omega A_p(k,\omega) = \pi\langle\left\{\hat{c}_k\hat{c}_k^\dagger\right\}\rangle = \pi \tag{9}$$

Using the `Max_Sac.f90` with `Channel="P"` will load the above Kernel in the MaxEnt library. Note that in this case the back transformation is set to unity.

3.5.3 Particle-hole quantities

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

$$\langle \hat{S}(q,\tau)\hat{S}(-q,0)\rangle = \frac{1}{\pi}\int \mathrm{d}\omega \frac{e^{-\tau\omega}}{1-e^{-\beta\omega}}\chi''(q,\omega). \tag{10}$$

This follows directly from the Lehmann representation:

$$\chi''(q,\omega) = \frac{\pi}{Z}\sum_{n,m} e^{-\beta E_n}|\langle n|\hat{S}(q)|\rangle m|^2\delta(\omega+E_n-E_m)\left(1-e^{-\beta\omega}\right) \tag{11}$$

Since the linear response to a Hermitian perturbation is real, $\chi''(q,\omega) = -\chi''(-q,-\omega)$. Hence for systems with inversion symmetry – that we will consider here – $\langle \hat{S}(q,\tau)\hat{S}(-q,0)\rangle$ is a symmetric function around $\beta = \tau/2$. The analysis file `cov_tau_ph.f90` produced at compilation time will use this to define an improved estimator.

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

$$\coth(\beta\omega/2)\chi''(q,\omega) \tag{12}$$

For this quantitiy, we have the sum rule since:

$$\int \mathrm{d}\omega \coth(\beta\omega/2)\chi''(q,\omega) = 2\pi\langle \hat{S}(q,\tau=0)\hat{S}(-q,0)\rangle \tag{13}$$

which is just the first point in the data.

Hence,

$$\langle \hat{S}(q,\tau)\hat{S}(-q,0)\rangle = \int \mathrm{d}\omega \underbrace{\frac{1}{\pi}\frac{e^{-\tau\omega}}{1-e^{-\beta\omega}}\tanh(\beta\omega/2)}_{K_{pp}(\tau,\omega)} \underbrace{\coth(\beta\omega/2)\chi''(q,\omega)}_{A(\omega)} \tag{14}$$

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

$$\langle \hat{S}(q,\tau)\hat{S}(-q,0)\rangle = \int_0^\infty \mathrm{d}\omega \underbrace{\left(K(\tau,\omega)+K(\tau,-\omega)\right)}_{K_{ph}(\tau,\omega)} A(\omega). \tag{15}$$

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

$$S(q,\omega) = \chi''(q,\omega)/\left(1-e^{-\beta\omega}\right). \tag{16}$$

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

$$S(q,\omega) = \frac{A(\omega)}{1+e^{-\beta\omega}} \tag{17}$$

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

$$\chi(q,i\Omega_m) = \int_0^\beta \mathrm{d}\tau e^{i\Omega_m\tau}\langle S(q,\tau)S(-q,0)\rangle = \frac{1}{\pi}\int \mathrm{d}\omega \frac{\chi''(q,\omega)}{\omega - i\Omega_m}. \tag{18}$$

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

$$\begin{aligned}\chi(q,i\Omega_m) &= \frac{1}{\pi}\int_0^\infty \mathrm{d}\omega\left(\frac{1}{\omega-i\Omega_m}-\frac{1}{-\omega-i\Omega_m}\right)\chi''(q,\omega)\\ &= \frac{2}{\pi}\int_0^\infty \mathrm{d}\omega\frac{\omega^2}{\omega^2+\Omega_m^2}\frac{\chi''(q,\omega)}{\omega} \equiv \int_0^\infty \mathrm{d}\omega K(\omega,i\Omega_m)A(q,\omega)\end{aligned} \tag{19}$$

with

$$K(\omega, i\Omega_m) = \frac{\omega^2}{\omega^2 + \Omega_m^2} \tag{20}$$

and

$$A(q,\omega) = \frac{2}{\pi}\frac{\chi''(q,\omega)}{\omega} \tag{21}$$

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

$$\int_0^\infty \mathrm{d}\omega A(q,\omega) = \frac{1}{\pi}\int_{-\infty}^{\infty}\mathrm{d}\omega\frac{\chi''(q,\omega)}{\omega} \equiv \chi(q, i\Omega_m = 0) \tag{22}$$

3.5.4 Particle-Particle quantities

Similarly to the particle-hole channel the particle-particle channel is also a bosonic correlation function. Here however we do not assume the imaginary time data is symmetric around the $\tau = \beta/2$ point. We use the Kernel K_{pp} define in Eq. 14 and consider the whole frequency range. The back transformation yields

$$\frac{\chi''(\omega)}{\omega} = \frac{\tanh(\beta\omega/2)}{\omega}A(\omega) \tag{23}$$

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 they wish to compute. In this section, we describe the module `Hamiltonian_Examples.f90` which contains an implementation of the Hubbard model on the square lattice. A sample run for this model can be found in `Examples/Hubbard_SU2_Square/`.

The Hamiltonian reads

$$\mathcal{H} = \sum_{\sigma=1}^{2}\sum_{x,y=1}^{N_{\text{unit cell}}} c^\dagger_{x\sigma}T_{x,y}c_{y\sigma} + \frac{U}{2}\sum_x\left[\sum_{\sigma=1}^{2}\left(c^\dagger_{x\sigma}c_{x\sigma} - 1/2\right)\right]^2 . \tag{24}$$

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 $M_I = 0$.

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. 3.1.1.

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 $N_{\text{dim}} \equiv N_{\text{unit cell}} \cdot$ `NORB` $=$ `Latt%N` $\cdot$ `NORB`, since $N_{\text{unit cell}} =$ `Latt%N`.

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 **??**. 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 main program will call the routine `ObserT(NT, GT0,G0T,G00,GTT, PHASE)` which is again provided by the user and handles imaginary-time displaced correlation functions.

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

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,

$$\texttt{GR(x,y,}\sigma\texttt{)} = \langle c_{x,\sigma} c^{\dagger}_{y,\sigma} \rangle, \tag{25}$$

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

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

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

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

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

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

$$\langle n_{\vec{i},\alpha} n_{\vec{j},\beta} \rangle - \langle n_{\vec{i},\alpha} \rangle \langle n_{\vec{j},\beta} \rangle \ . \tag{26}$$

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

$$\texttt{GRC(x,y,s)} = \delta_{x,y} - \texttt{GR(y,x,s)} \tag{27}$$

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

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

Note that we consider the square lattice of the single site Hubbard model as a special case of a multi-orbital problem as described in Sec. 4.3.1 At the end of each bin the main program will call the routine `Pr_obs(LTAU)`. This routine will append the result of the bins in the specified file, with appropriate suffix.

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:

$$\begin{aligned}
\texttt{GT0(x,y,s)} &= \langle\langle \hat{c}_{x,s}(Nt\Delta\tau)\hat{c}^{\dagger}_{y,s}(0)\rangle\rangle = \langle\langle \mathcal{T}\hat{c}_{x,s}(Nt\Delta\tau)\hat{c}^{\dagger}_{y,s}(0)\rangle\rangle \\
\texttt{G0T(x,y,s)} &= -\langle\langle \hat{c}^{\dagger}_{y,s}(Nt\Delta\tau)\hat{c}_{x,s}(0)\rangle\rangle = \langle\langle \mathcal{T}\hat{c}_{x,s}(0)\hat{c}^{\dagger}_{y,s}(Nt\Delta\tau)\rangle\rangle \\
\texttt{G00(x,y,s)} &= \langle\langle \hat{c}_{x,s}(0)\hat{c}^{\dagger}_{y,s}(0)\rangle\rangle \\
\texttt{GTT(x,y,s)} &= \langle\langle \hat{c}_{x,s}(Nt\Delta\tau)\hat{c}^{\dagger}_{y,s}(Nt\Delta\tau)\rangle\rangle
\end{aligned} \tag{28}$$

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:

$$4\langle\langle \hat{S}^z_{\vec{i}}(\tau)\hat{S}^z_{\vec{j}}(0)\rangle\rangle = -2\ \texttt{G0T(J1,I1,1) GT0(I1,J1,1)} \quad . \tag{29}$$

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.

4.1.3 Numerical precision

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

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

showing the mean and maximum difference between the *wrapped* and from scratched computed equal and time displaced Green functions [4]. A stable code should produce results where the mean difference is smaller than the stochastic error. The above example shows a very stable simulation since the Green function is of order one.

4.2 The M_z-Hubbard model on a square lattice

The Hubbard Hamiltonian can equally be written as:

$$\mathcal{H} = \sum_{\sigma=1}^{2}\sum_{x,y=1}^{N_{unit\ cells}} c^{\dagger}_{x\sigma}T_{x,y}c_{y\sigma} - \frac{U}{2}\sum_{x}\left[c^{\dagger}_{x,\uparrow}c_{x\uparrow} - c^{\dagger}_{x,\downarrow}c_{x\downarrow}\right]^2 \quad . \tag{30}$$

We can make contact with the general form of the Hamiltonian (see Eq. **??**) by setting: $N_{\rm fl} = 2$, $N_{\rm col} \equiv$ `N_SUN` $= 1$, $M_T = 1$, $T^{(ks)}_{xy} = T_{x,y}$, $M_V = N_{\text{unit cell}}$, $U_k = \frac{U}{2}$, $V^{(k,s=1)}_{xy} = \delta_{x,y}\delta_{x,k}$, $V^{(k,s=2)}_{xy} = -\delta_{x,y}\delta_{x,k}$, $\alpha_{ks} = 0$ and $M_I = 0$. The coupling of the HS fields to the z-component of the magnetization breaks the $SU(2)$ spin symmetry. Nevertheless the z-component of the spin remains a good quantum number such that the imaginary-time propagator – for a given HS field – is block diagonal in this quantum number. This corresponds to the flavor index which runs from one to two labelling spin up and spin down degrees of freedom. In the parameter file listed in Sec. 3.1.1 setting the model variable to `Hubbard_Mz` will carry out the simulation in the above representation. With respect to the $SU(2)$ case, the changes required in the `Hamiltonian_Examples.f90` module are minimal and essentially effect only the interaction term, and the calculation of observables. We note that in this formulation the hopping matrix can be flavor dependent such that a Zeeman magnetic field can be introduced. If the chemical potential is set to zero, this will not generate a negative sign problem [7, 8, 9]. 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 $4\langle\langle \hat{S}^z_i \hat{S}^z_j \rangle\rangle$ calculated in the subroutine `Obser` will take the form:

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

Here, `GRC` is defined in Eq. 27. 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.

4.2.3 Numerical precision

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

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

This is still an excellent precision but nevertheless choosing a HS field which couples to the z-component of the magnetization apparently leads to numerical results that are a couple of order of magnitudes less precise than a HS decomposition coupling to the charge (compare with Sec. 4.1.3).

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

The Hamilton module `Hamiltonian_Examples.f90` can also carry out simulations for the Hubbard model on the Honeycomb lattice by setting in the parameter file `Lattice_type="Honeycomb"` (see Sec. 3.1.1). 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 to implement this in the code. The Honeycomb lattice is a triangular Bravais lattice with two orbitals per unit cell. The routine `Ham_Latt` will set:

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

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

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

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

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 the chemical potential
      I1 = invlist(I,no)
      Op_T(nc,n)%O(I1 ,I1) = cmplx(-Ham_chem, 0.d0, kind(0.D0))
   enddo
   I1 = Invlist(I,1)                             ! Orbital A of unit cell I
   Do nc1 = 1,N_coord                            ! Loop over coordination  number
      select case (nc1)
      case (1)
         J1 = invlist(I,2)                       ! Orbital B of unit cell i
      case (2)
         J1 = invlist(Latt%nnlist(I,1,-1),2)     ! Orbital B of unit cell i + a_1 - a_2
      case (3)
         J1 = invlist(Latt%nnlist(I,0,-1),2)     ! Orbital B of unit cell i - a_2
```

```
      case default
         Write(6,*) ' Error in  Ham_Hop '
      end select
      Op_T(nc,n)%O(I1,J1) = cmplx(-Ham_T,    0.d0, kind(0.D0))
      Op_T(nc,n)%O(J1,I1) = cmplx(-Ham_T,    0.d0, kind(0.D0))
   Enddo
Enddo
```

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

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 have already considered the Hubbard model on the square lattice to correspond to a multi-orbital unit cell albeit with the special choice of one orbital per unit cell.

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:

$$\begin{aligned}\mathcal{H} = & \sum_{\sigma=1}^{2}\sum_{x,y} c^{\dagger}_{x\sigma}T_{x,y}c_{y\sigma} + \frac{U}{2}\sum_{x}\left[\sum_{\sigma=1}^{2}\left(c^{\dagger}_{x\sigma}c_{x\sigma}-1/2\right)\right]^2 + \xi\sum_{\sigma,\langle x,y\rangle}\hat{Z}_{\langle x,y\rangle}\left(c^{\dagger}_{x\sigma}c_{y\sigma}+h.c.\right) \\ & -h\sum_{\langle x,y\rangle}\hat{X}_{\langle x,y\rangle} - J\sum_{\langle\langle x,y\rangle\langle x',y'\rangle\rangle}\hat{Z}_{\langle x,y\rangle}\hat{Z}_{\langle x',y'\rangle} \end{aligned} \tag{31}$$

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

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

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

4.4.1 The Ising term.

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

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

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

4.4.2 The interaction term: `Call Ham_V`

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

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

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

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

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

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

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

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

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

$$C = \{s_{i,\tau}, l_{j,\tau} \text{ with } i = 1 \cdots M_I, j = 1 \cdots M_V, \tau = 1, L_{Trotter}\} \ . \tag{32}$$

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

however `Op_V(n,1)%type = 1` then function `S0` returns

$$\frac{e^{-S_{0,I}\left(s_{1,\tau},\cdots,-s_{n,\tau},\cdots s_{M_I,\tau}\right)}}{e^{-S_{0,I}\left(s_{1,\tau},\cdots,s_{n,\tau},\cdots s_{M_I,\tau}\right)}} \tag{33}$$

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

5 Miscellaneous

5.1 Other models

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

5.1.1 Kondo lattice model

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

$$\hat{\mathcal{H}} = \underbrace{-t \sum_{\langle \vec{i},\vec{j}\rangle,\sigma} \left(\hat{c}^\dagger_{\vec{i},\sigma}\hat{c}_{\vec{j},\sigma} + \text{H.c.}\right)}_{\equiv\hat{\mathcal{H}}_t} - \frac{J}{4}\sum_{\vec{i}}\left(\sum_\sigma \hat{c}^\dagger_{\vec{i},\sigma}\hat{f}_{\vec{i},\sigma} + \hat{f}^\dagger_{\vec{i},\sigma}\hat{c}_{\vec{i},\sigma}\right)^2 + \underbrace{\frac{U}{2}\sum_{\vec{i}}\left(\hat{n}^f_{\vec{i}} - 1\right)^2}_{\equiv\hat{\mathcal{H}}_U}. \tag{34}$$

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

$$\hat{\mathcal{H}} = \hat{\mathcal{H}}_t + J\sum_{\vec{i}}\left(\hat{\vec{S}}^c_{\vec{i}}\cdot\hat{\vec{S}}^f_{\vec{i}} + \hat{\eta}^{z,c}_{\vec{i}}\cdot\hat{\eta}^{z,f}_{\vec{i}} - \hat{\eta}^{x,c}_{\vec{i}}\cdot\hat{\eta}^{x,f}_{\vec{i}} - \hat{\eta}^{y,c}_{\vec{i}}\cdot\hat{\eta}^{y,f}_{\vec{i}}\right) + \hat{\mathcal{H}}_U. \tag{35}$$

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

$$\hat{\eta}^\alpha_{\vec{i}} = \hat{P}^{-1}\hat{S}^\alpha_{\vec{i}}\hat{P} \text{ with } \hat{P}^{-1}\hat{c}_{\vec{i},\uparrow}\hat{P} = (-1)^{i_x+i_y}\hat{c}^\dagger_{\vec{i},\uparrow} \text{ and } \hat{P}^{-1}\hat{c}_{\vec{i},\downarrow}\hat{P} = \hat{c}_{\vec{i},\downarrow} \tag{36}$$

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,

$$\left[\hat{\mathcal{H}}, \hat{\mathcal{H}}_U\right] = 0. \tag{37}$$

Thereby, and for positive values of U , doubly occupied or empty f-sites – corresponding to even parity sites – are suppressed by a Boltzmann factor $e^{-\beta U/2}$ in comparison to odd parity sites. Choosing βU adequately essentially allows to restrict the Hilbert space to odd parity f-sites. In this Hilbert space $\hat{\eta}^{x,f} = \hat{\eta}^{y,f} = \hat{\eta}^{z,f} = 0$ such that the Hamiltonian (34) reduces to the Kondo lattice model.

5.1.2 $SU(N)$ Hubbard-Heisenberg models

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

$$\hat{\mathcal{H}} = \underbrace{-t\sum_{\langle\vec{i},\vec{j}\rangle}\left(\vec{\hat{c}}^\dagger_{\vec{i}}\vec{\hat{c}}_{\vec{j}} + \text{H.c.}\right)}_{\equiv\hat{\mathcal{H}}_t} \underbrace{-\frac{J}{2N}\sum_{\langle\vec{i},\vec{j}\rangle}\left(\hat{D}^\dagger_{\vec{i},\vec{j}}\hat{D}_{\vec{i},\vec{j}} + \hat{D}_{\vec{i},\vec{j}}\hat{D}^\dagger_{\vec{i},\vec{j}}\right)}_{\equiv\hat{\mathcal{H}}_J} + \underbrace{\frac{U}{N}\sum_{\vec{i}}\left(\vec{\hat{c}}^\dagger_{\vec{i}}\vec{\hat{c}}_{\vec{i}} - \frac{N}{2}\right)^2}_{\equiv\hat{\mathcal{H}}_U} \tag{38}$$

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 QMC code of the ALF project to simulate this model, one will rewrite the J-term as a sum of perfect squares,

$$\hat{\mathcal{H}}_J = -\frac{J}{4N}\sum_{\langle\vec{i},\vec{j}\rangle}\left(\hat{D}^\dagger_{\langle\vec{i},\vec{j}\rangle} + \hat{D}_{\langle\vec{i},\vec{j}\rangle}\right)^2 - \left(\hat{D}^\dagger_{\langle\vec{i},\vec{j}\rangle} - \hat{D}_{\langle\vec{i},\vec{j}\rangle}\right)^2, \tag{39}$$

so to manifestly bring it into the form of the general Hamiltonian(**??**). It is amusing to note that setting the hopping $t = 0$, charge fluctuations will be suppressed by the Boltzmann factor $e^{-\beta U/N\left(\hat{\vec{c}}_{\vec{i}}^{\dagger}\hat{\vec{c}}_{\vec{i}}-\frac{N}{2}\right)^2}$ since in this case $\left[\hat{\mathcal{H}}_J, \hat{\mathcal{H}}_U\right] = 0$. This provides a route to use the auxiliary field QMC algorithm to simulate – free of the sign problem – $SU(2N)$ Heisenberg models in the self-adjoint antisymmetric representation [3] For odd values of N recent progress in our understanding of the origins of the sign problem [18] allows us to simulate a set of non-trivial Hamiltonians [19, 12], without encountering the sign problem.

5.1.3 Hubbard model in the canonical ensemble

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

$$\hat{\mathcal{H}} = \hat{\mathcal{H}_{tU}} + \underbrace{\lambda\left(\hat{N} - N\right)^2}_{\equiv \hat{H}_\lambda} \tag{40}$$

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

$$S(\vec{q} = 0) = \sum_{\vec{r}} \left[\langle \hat{n}_{\vec{r}} \hat{n}_{\vec{0}} \rangle - \langle \hat{n}_{\vec{r}} \rangle \langle \hat{n}_{\vec{0}} \rangle\right] \tag{41}$$

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

$$e^{-\beta \hat{H}} = \prod_{\tau=1}^{L_{\mathrm{Trotter}}} \left[e^{-\Delta\tau \hat{H}_t} e^{-\Delta\tau \hat{H}_U} \underbrace{e^{-\frac{\Delta\tau}{n_\lambda}\hat{H}_\lambda} \cdots e^{-\frac{\Delta\tau}{n_\lambda}\hat{H}_\lambda}}_{n_\lambda\text{-times}} \right]. \tag{42}$$

Thereby, we need n_λ fields per time slice to impose the constraint. Since for each field the coupling constant is suppressed by a factor n_λ, we can monitor the acceptance. An implementation of this program can be found in `Prog/Hamiltonian_Hub_Canonical.f90` and a test run in the directory `Examples/Hubbard_Mz_Square_Can`

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

5.1.4 Z_2 slave spin formualtion of the Hubbard model

In this subsection, we demonstrate that the code can be used to simulate the attractive Hubbard model in the Z_2-slave spin formulation [20].

$$\hat{H} = -t \sum_{\langle \vec{i},\vec{j}\rangle,\sigma} \hat{c}^{\dagger}_{\vec{i},\sigma} \hat{c}_{\vec{j},\sigma} - U \sum_{i} \left(\hat{n}_{\vec{i},\uparrow} - 1/2\right)\left(\hat{n}_{\vec{i},\downarrow} - 1/2\right) \tag{43}$$

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

$$\hat{c}^{\dagger}_{\vec{i},\sigma} = \hat{\tau}^{z}_{\vec{i}} \hat{f}^{\dagger}_{\vec{i},\sigma}. \tag{44}$$

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

$$\hat{\tau}^{x}_{i} - (-1)^{\sum_{\sigma} \hat{f}^{\dagger}_{i,\sigma} \hat{f}_{i,\sigma}} = 0 \tag{45}$$

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

$$\hat{Q}_{\vec{i}} = \tau^{x}_{\vec{i}} (-1)^{\sum_{\sigma} \hat{f}^{\dagger}_{\vec{i},\sigma} \hat{f}_{\vec{i},\sigma}} = 1. \tag{46}$$

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

$$\hat{H}_{Z_2} = -t \sum_{\langle \vec{i},\vec{j}\rangle,\sigma} \hat{\tau}^{z}_{\vec{i}} \hat{\tau}^{z}_{\vec{j}} \hat{f}^{\dagger}_{\vec{i},\sigma} \hat{f}_{\vec{j},\sigma} - \frac{U}{4} \sum_{\vec{i}} \hat{\tau}^{x}_{\vec{i}} \tag{47}$$

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

$$\left[\hat{H}_{Z_2}, \hat{Q}_{\vec{i}}\right] = 0. \tag{48}$$

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

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

$$\hat{Z}_{\langle \vec{i},\vec{j}\rangle} = \hat{\tau}^{z}_{\vec{i}} \hat{\tau}^{z}_{\vec{j}} \tag{49}$$

such that

$$\hat{\tau}^{x}_{\vec{i}} = \hat{X}_{\vec{i},\vec{i}+\vec{a}_x} \hat{X}_{\vec{i},\vec{i}-\vec{a}_x} \hat{X}_{\vec{i},\vec{i}+\vec{a}_y} \hat{X}_{\vec{i},\vec{i}-\vec{a}_y}. \tag{50}$$

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

$$\hat{Z}_{\vec{i},\vec{i}+\vec{a}_x} \hat{Z}_{\vec{i}+\vec{a}_x,\vec{i}+\vec{a}_x+\vec{a}_y} \hat{Z}_{\vec{i}+\vec{a}_x+\vec{a}_y,\vec{i}+\vec{a}_y} \hat{Z}_{\vec{i}+\vec{a}_y,\vec{i}} = 1 \quad \forall \quad \vec{i}. \tag{51}$$

Within this formulation the model takes the form:

$$\hat{H}_{Z_2} = -t \sum_{\langle \vec{i},\vec{j}\rangle,\sigma} \hat{Z}_{\langle \vec{i},\vec{j}\rangle} \hat{f}^{\dagger}_{\vec{i},\sigma} \hat{f}_{\vec{j},\sigma} - \frac{U}{4} \sum_{\vec{i}} \hat{X}_{\vec{i},\vec{i}+\vec{a}_x} \hat{X}_{\vec{i},\vec{i}-\vec{a}_x} \hat{X}_{\vec{i},\vec{i}+\vec{a}_y} \hat{X}_{\vec{i},\vec{i}-\vec{a}_y}. \tag{52}$$

The fermion part has formally the same from as in the Hubbard model coupled to a dynamical Ising field discussed in Sec. 4.4. There are however two important differences.

- The moves have to respect the constraint of Eq. 51. Thereby single spin flip terms are prohibited and the minimal move one can carry out on a given time slice is the following. We randomly choose a site $\vec{i}$ and propose a move where: $Z_{\vec{i},\vec{i}+\vec{a}_x} \to -Z_{\vec{i},\vec{i}+\vec{a}_x}$, $Z_{\vec{i},\vec{i}-\vec{a}_x} \to -Z_{\vec{i},\vec{i}-\vec{a}_x}$, $Z_{\vec{i},\vec{i}+\vec{a}_y} \to -Z_{\vec{i},\vec{i}+\vec{a}_y}$ and $Z_{\vec{i},\vec{i}-\vec{a}_y} \to -Z_{\vec{i},\vec{i}-\vec{a}_y}$. One can carry out such moves by using the global move in real space option presented in Secs. **??** and 3.1.1.

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

We have carried out some test simulations at half-filling and at low temperatures. The simulation can be found in directory `Examples/Z2_Slave` and the Hamiltonian in `Hamiltonian_Z2_slave_spins.f90` . The simulations are carried out at half-filling such that particle-hole symmetry leads to

$$\langle\hat{Q}_{\vec{i}}\rangle_{H_{Z_2}} = 0. \tag{53}$$

However the simulations suggest that

$$\frac{1}{N}\sum_{i,j}\langle\hat{Q}_{\vec{i}}\hat{Q}_{\vec{j}}\rangle_{H_{Z_2}} = N \tag{54}$$

at low temperatures where N corresponds to size of the lattice. Note that this measurement is very noisy, and suffers from very long autocorrelation times. Thereby, the constraint is dynamically imposed and simulations of the attractive Hubbard model with `Hamiltonian_Examples.f90` should yield identical results. To test this, we have computed equal time Green functions:

$$\langle\hat{\tau}^z_{\vec{i}}\hat{f}^\dagger_{\vec{i},\sigma}\hat{\tau}^z_{\vec{j}}\hat{f}_{\vec{j},\sigma}\rangle_{H_{Z_2}} = \langle\hat{c}^\dagger_{\vec{i},\sigma}\hat{c}_{\vec{j},\sigma}\rangle_H \tag{55}$$

as obtained from the `Hamiltonian_Examples.f90` and `Hamiltonian_Z2_slave_spins.f90` codes. A test run for the 8×8 lattice at $U/t = 4$ and $\beta t = 40$ gives:

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

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

5.1.5 Z_2 gauge theory coupled to Z_2 matter.

The Hamiltonian we will consider here reads

$$\begin{aligned}\hat{H} &= -t_{Z_2}\sum_{\langle\vec{i},\vec{j}\rangle,\sigma}\hat{Z}_{\langle\vec{i},\vec{j}\rangle}\left(\hat{\Psi}^\dagger_{\vec{i},\sigma}\hat{\Psi}_{\vec{j},\sigma}+h.c.\right)-\mu\sum_{\vec{i},\sigma}\hat{\Psi}^\dagger_{\vec{i},\sigma}\hat{\Psi}_{\vec{i},\sigma}-g\sum_{\langle\vec{i},\vec{j}\rangle}\hat{X}_{\langle\vec{i},\vec{j}\rangle}+K\sum_{\square}\prod_{\langle\vec{i},\vec{j}\rangle\in\partial\square}\hat{Z}_{\langle\vec{i},\vec{j}\rangle}\\ &+J\sum_{\langle\vec{i},\vec{j}\rangle}\hat{\tau}^z_{\boldsymbol{i}}\hat{Z}_{\langle\vec{i},\vec{j}\rangle}\hat{\tau}^z_{\boldsymbol{i}}-h\sum_{\vec{i}}\hat{\tau}^x_{\vec{i}}-t\sum_{\langle\vec{i},\vec{j}\rangle,\sigma}\hat{\tau}^z_{\boldsymbol{i}}\hat{\tau}^z_{\boldsymbol{j}}\left(\hat{\Psi}^\dagger_{\vec{i},\sigma}\hat{\Psi}_{\vec{j},\sigma}+h.c.\right)\end{aligned} \tag{56}$$

5.1.6 Long range Coulomb

The model we consider here reads:

$$\hat{H}=-t\sum_{\vec{i},\vec{j},\sigma=1}^{N}\hat{c}^\dagger_{\vec{i},\sigma}T_{\vec{i},\vec{j}}\hat{c}_{\vec{j},\sigma}+\frac{1}{N}\sum_{\vec{i},\vec{j}}\left(\hat{n}_{\vec{i}}-\frac{N}{2}\right)V_{\vec{i},\vec{j}}\left(\hat{n}_{\vec{j}}-\frac{N}{2}\right)\quad\text{with }\hat{n}_{\vec{i}}=\sum_{\sigma=1}^{N}\hat{c}^\dagger_{\vec{i},\sigma}\hat{c}_{\vec{i},\sigma} \tag{57}$$

The interaction is specified in the following way:

$$V_{\vec{i},\vec{j}}=U\begin{cases}1 & \text{if } \vec{i}-\vec{j}=0\\ \frac{\alpha\, d_{min}}{|\vec{i}-\vec{j}|} & \text{otherwise}\end{cases} \tag{58}$$

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

$$e^{-\Delta\tau\hat{H}_V}=\int\prod_{\vec{i}}d\phi_{\vec{i}}e^{-\frac{N\Delta\tau}{4}\phi_{\boldsymbol{i}}V^{-1}_{\boldsymbol{i},\boldsymbol{j}}\phi_{\boldsymbol{j}}-\sum_{\boldsymbol{i}}i\Delta\tau\phi_i\left(n_i-\frac{N}{2}\right)} \tag{59}$$

The implementation follows Ref. [21] but now supports various lattice geometries. The definition of the Coulomb repulsion is as follows. A general lattice site `I,n` where `I: 1...Latt%N` is the unit cell and `n = 1 ...Latt_unit%NORB` the orbital is given by:

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

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

$$V(\vec{i}+\vec{\delta}_i,\vec{j}+\vec{\delta}_j)=\frac{Ud_{min}\alpha}{|\overline{\vec{i}-\vec{j}}+\vec{\delta}_i-\vec{\delta}_j|} \tag{60}$$

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

5.2 Performance, memory requirements and parallelization

As mentioned in the introduction, the auxiliary field QMC algorithm scales linearly in inverse temperature β and cubic in the volume N_{dim}. Using fast updates, a single spin flip requires $(N_{\mathrm{dim}})^2$ operations to update the Green function upon acceptance. As there are $L_{\mathrm{Trotter}}\times N_{\mathrm{dim}}$ spins to be visited, the total computational cost for one sweep is of the order of $\beta(N_{\mathrm{dim}})^3$. This operation dominates the performance, see Fig. 1. A profiling analysis of our code shows that 80-90% of the CPU time is spend in ZGEMM calls of the BLAS library provided in the MKL package by Intel. Consequently, the single-core performance is next to optimal.

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

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

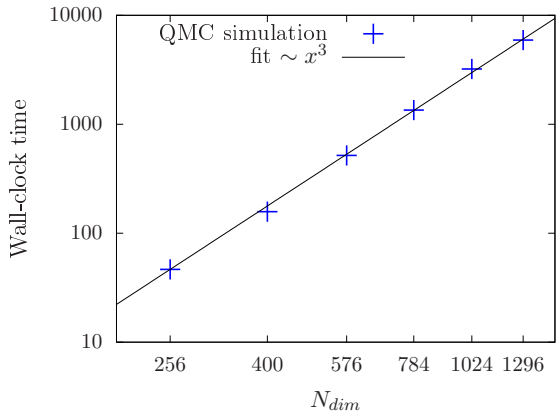

Figure 1: Volume scaling behavior of the auxiliary field QMC code of the ALF project on SuperMUC (phase 2/Haswell nodes) at the LRZ in Munich. The number of sites N_{dim} corresponds to the system volume. The plot confirms that the leading scaling order is due to matrix multiplications such that the runtime is dominated by calls to ZGEMM.

$T_{\mathrm{auto}}/N_{\mathrm{OMP}}$, where N_{MPI} is the number of cores and N_{OMP} the number of OpenMP threads. For a given number of independent measurements N_{meas}, we therefore need a wall-clock time given by

$$T = \frac{T_{\mathrm{auto}}}{N_{\mathrm{OMP}}} \left(1 + \frac{N_{\mathrm{meas}}}{N_{\mathrm{MPI}}} \right) . \tag{61}$$

As we typically have $N_{\mathrm{meas}}/N_{\mathrm{MPI}} \gg 1$, the speedup is expected to be almost perfect, in accordance with the performance test results for the auxiliary field QMC code on SuperMUC (see Fig. 2 (left)).

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

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

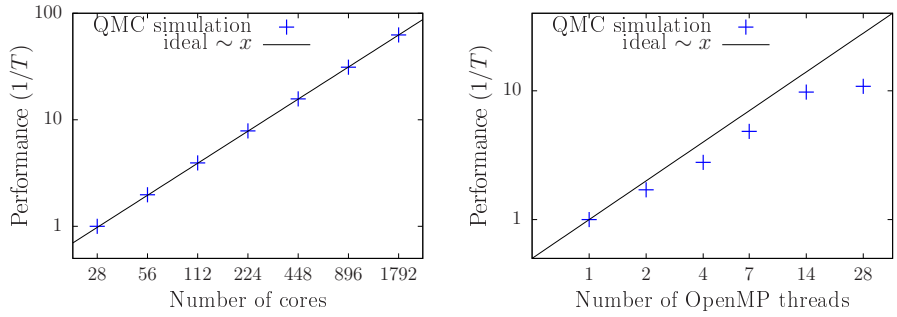

Figure 2: MPI (left) and OpenMP (right) scaling behavior of the auxiliary field QMC code of the ALF project on SuperMUC (phase 2/Haswell nodes) at the LRZ in Munich. The MPI performance data was normalized to 28 cores and was obtained using a problem size of $N_{\mathrm{dim}} = 400$. This is a medium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

6 Conclusions and Future Directions

In its present form, the auxiliary field QMC code of the ALF project allows to simulate a large class of non-trivial models, both efficiently and at minimal programming cost. There are many possible extensions which deserve to be considered in future releases. The model Hamiltonians we have presented so far are imaginary-time independent. This however can be easily generalized to imaginary-time dependent model Hamiltonians thus allowing, for example, to access entanglement properties of interacting fermionic systems [22, 23, 24, 25]. Generalizations to include global moves are equally desirable. This is a prerequisite to play with recent ideas of self-learning algorithms [26] 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 [21, 27, 28] – are not yet included. Extensions to continuous HS fields are certainly possible, but require an efficient upgrading scheme. Finally, an implementation of the ground state projective QMC method is equally desirable.

Acknowledgments

We are very grateful to S. Beyl, M. Hohenadler, F. Parisen Toldin, M. Raczkowski, T. Sato, J. Schwab, Z. Wang, and M. Weber for constant support during the development of this project. MB: I think we should acknowledge also RRZE Erlangen: We equally thank G. Hager, M. Wittmann, and G. Wellein for useful discussions and support. FFA would also like to thank T. Lang and Z. Y. Meng for developments of the auxiliary field code as well as T. Grover. MB thanks the Bavarian Competence Network for Technical and Scientific High Performance Computing (KONWIHR) for financial support. FG and JH thank the SFB-1170 for financial support under projects Z03 and C01. FFA thanks the DFG-funded FOR1807 and FOR1346 for partial financial support. Part of the optimization of the code was carried out during the Porting and Tuning Workshop 2016 offered by the Forschungszentrum Jülich. Calculations to extensively test this package were carried out both on SuperMUC at the Leibniz Supercomputing Centre and on JURECA [29] at the Jülich Supercomputing Centre. We thank both institutions for generous allocation of computing time.

References

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

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

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

[4] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

[28] R. Brower, C. Rebbi, and D. Schaich, PoS **LATTICE2011**, 056 (2011).

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

License

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

The ALF project's contributors.

COPYRIGHT