# Documentation for the General QMC code

Martin Bercx

September 22, 2016

## Using the code

*Example simulation, tutorial: where to find and how to start*

### Parameter files

*describe the input parameters, give sample values for the stabilization parameters*

### Analysis files

*how the analysis of Monte Carlo data is done*

# List of files

*all files that constitute the code, with a brief description*

## cgr1.f90 & cgr2.f90

Stable computation of the physical single-particle equal time Green function $G(\tau)$.

### control_mod.f90

Includes a set of auxiliary routines, regarding the flow of the simulation. Examples are initialization of performance variables, precision tests and controlled termination of the code.

**gperp.f90**

### Hamiltonian_Hub.f90

Here, the physical simulation parameters (the model parameters) and the lattice parameters are read in. The lattice, the non-interacting and the interacting part of the Hubbard Hamiltonian are set according to the parameters and the chosen Hubbard-Stratonovich decomposition.

**Hop_mod.f90**

### inconfc.f90

The auxiliary-field QMC method is based on a Hubbard-Stratonovich decomposition of the interaction term. This decomposition introduces a space-time array of (discrete) configurations of auxiliary fields, i.e. Ising spins. In this routine, an existing configuration is read in, checks on its dimensionality are made and, in case no prior configuration exists, a random configuration of Ising spins is set up.

## main.f90

Top-level part of the program. Here, the program flow which consists of initialization, sweeps through the space-time lattice, and finalizing the program, is coded.

### nranf.f90

Auxiliary routine controlling the evaluation of random numbers.

### Operator.f90

The algorithms is centered around evaluation of single-particle operators, represented as square matrices. In this routine, the abstract type Operator is defined, including information on the coupling strength, the sites that participate in the single-particle hopping process, and the type of Hubbard-Stratonovich transformation. This routine collects all program relevant operations that are applied to the type Operator, like initializations or multiplications.

## outconfc.f90

```
Description in plain text:
At the end of the simulation, the last configuration of Hubbard-Stratonovich variables,
together with the last set of random numbers  is written to the file confout.
Prior to the start of a new simulation of the identical space-time dimesion,
one can (manually) copy the file confout to the file confin and
make the new run use the old configuration.
Doing this saves warmup time compared to a complete random (unphysical) configuration.

Input/output variables
in:
inout:
inout:
out:

Dependencies
include: mpif.h
modules: Hamiltonian
interfaces:
global variables:  ntrot, nsigma
subroutines:  MPI_COMM_RANK, MPI_COMM_SIZE, Get_seed_Len, Ranget

Things to check:
Rename subroutine to confout.f90 for consistency
```

### print_bin_mod.f90

```
Description in plain text:
Here the way to write the measure bins to the respective output files is coded.
A bin is an average over many individual measurements.
The bin defines the unit of Monte Carlo time.

Dependencies
include:
modules:
interfaces: Print_bin

Contains
subroutines: Print_bin_c, Print_bin_r, Print_scal, Print_bin_tau

Things to check:
```

#### Print_bin_C

```
Description in plain text:

Input/output variables
in: Latt, type(Lattice)
in: Phase_bin_tmp, complex
in: File_pr, character(Len=64)
in: nobs, integer
inout: Dat_eq,(:,:,:), complex
inout: Dat_eq0(:), complex
out:

Dependencies
include: mpif.h
modules: Lattices_v3
interfaces:
global variables:  TYPE(LATTICE), N, listk, b1_p, b2_p
subroutines:  MPI_COMM_RANK, MPI_COMM_SIZE, MPI_REDUCE, Fourier_R_to_k

Things to check:
STATUS(MPI_STATUS_SIZE), integer,  needed?
```

#### Print_bin_R

```
Description in plain text:

Input/output variables
in: Latt, type(Lattice)
in: Phase_bin_tmp, complex
in: File_pr, character(Len=64)
in: nobs, integer
inout: Dat_eq,(:,:,:), real
inout: Dat_eq0(:), real
out:

Dependencies
include: mpif.h
```

```
modules: Lattices_v3
interfaces:
global variables:  TYPE(LATTICE), N, listk, b1_p, b2_p
subroutines:  MPI_COMM_RANK, MPI_COMM_SIZE, MPI_REDUCE, Fourier_R_to_k

Things to check:
STATUS(MPI_STATUS_SIZE), integer,  needed?
```

### Print_scal

```
Description in plain text:

Input/output variables
in: File_pr, character(Len=64)
in: nobs, integer
inout: Obs,(:), complex
out:

Dependencies
include: mpif.h
interfaces:
global variables:
subroutines:  MPI_COMM_RANK, MPI_COMM_SIZE, MPI_REDUCE

Things to check:
STATUS(MPI_STATUS_SIZE), integer,  needed?
change subroutine name to print_bin_scal for consistency
```

### Print_bin_tau

```
Description in plain text:

Input/output variables
in: Latt, type(Lattice)
in: Phase_bin, complex
in: File_pr, character(Len=64)
in: nobs, integer
in: dtau, real
inout: Dat_tau,(:,:,:,:), complex
inout, optional: Dat0_tau(:), complex
out:

Dependencies
include: mpif.h
modules: Lattices_v3
interfaces:
global variables:  TYPE(LATTICE), N, listk, b1_p, b2_p
subroutines:  MPI_COMM_RANK, MPI_COMM_SIZE, MPI_REDUCE, Fourier_R_to_k

Things to check:
STATUS(MPI_STATUS_SIZE), integer,  needed?
```

## tau_m.f90

```
Description in plain text:
module tau_m_mod, with several subroutines

Dependencies
include:
modules: Hamiltonian, Operator_mod, Precdef, Control, Hop_mod

Contains
subroutines: tau_m, propr, proprm1

Things to check:
change file name to tau_m_mod.f90 for consistency
```

### TAU_M

```
Description in plain text:

Input/output variables
in: nstm, integer
in: nwrap, integer
in: ust(ndim,ndim; nstm,n_fl), complex
in: vst(ndim,ndim,nstm,n_fl), complex
in: dst(ndim,nstm,n_fl), complex
in: GR(ndim,ndim,n_fl), complex
in: phase, complex
in: stab_nt(0:nstm), integer
inout:
inout:
out:

Dependencies
include:
modules:
interfaces: wrapul, cgr2_1, cgr2_2, cgr2
global variables:  ndim, n_fl, cone, ltrot
subroutines:  obsert, initd,propr, proprm1, wrapur, cgr2_2, Control_Precision

Things to check:
cone?
```

### propr

```
Description in plain text:

Input/output variables
in: nt
inout: Ain(ndim,ndim,n_fl)
inout:
out:

Dependencies
include:
modules:
```

interfaces:
global variables: ndim, n_fl, op_v, nsigma, Phi, type
subroutines: Hop_mod_mmthr, Op_mmultR

Things to check:

### proprm1

Description in plain text:

Input/output variables
in: nt
inout: Ain(ndim,ndim,n_fl)
inout:
out:

Dependencies
include:
modules:
interfaces:
global variables: ndim, n_fl, op_v, nsigma, Phi, type
subroutines: Hop_mod_mmthl, Op_mmultL

Things to check:

## UDV_WRAP.f90

```
Description in plain text:
UDV_Wrap_mod is a module file, containing subroutines on the stabilization of
matrix computations.

Dependencies
modules: MyMats, Files_mod

Contains
subroutines: UDV_Wrap_Pivot, UDV_Wrap

Things to check:
change file name to  UDV_Wrap_mod.f90 for consistency
```

### UDV_Wrap_Pivot

```
Description in plain text:

Input/output variables
in: A(:,:), complex
in: ncon, integer
in: n1, integer
in: n2, integer
inout: U(:,:), complex
inout: V(:,:), complex
inout: D(:), complex
out:

Dependencies
include:
modules:
interfaces:
global variables:
subroutines:  UDV_Wrap, MMULT, Compare

Things to check:
```

### UDV_Wrap

```
Description in plain text:

Input/output variables
in: A(:,:), complex
in: ncon, integer
inout: U(:,:), complex
inout: V(:,:), complex
inout: D(:), complex
out:

Dependencies
include: mpif.h
modules:
interfaces:
global variables:
```

```
subroutines:  MPI_COMM_SIZE, MPI_COMM_RANK, QR, SVD, MMULT

Things to check:
STATUS(MPI_STATUS_SIZE), integer: not used
```

### upgrade.f90

```
Description in plain text:
The update of the Hubbard-Stratonovich configuration is done sequentially
for each point in the space-time lattice,
i.e. one Hubbard-Stratonovich Ising spin after the other.
In this routine, an update (i.e. a spin flip) is accepted or rejected.
The decision is made using the Metropolis method of importance sampling.

Input/output variables
in: N_op, integer
in: nt, integer
in: OP_dim, integer
inout: GR(ndim,ndim,n_fl), complex
inout: Phase, complex
out:

Dependencies
include:
modules: Hamiltonian, Random_wrap, Control, Precdef
interfaces:
global variables: ndim, n_fl, op_v, nflipl, Phi, n_non_zero, Gaml, P, nsigma, g, alpha, type, E
subroutines:  zgemm, control_upgrade

Things to check:
nranf, integer, external (where is external fct. defined)
log, logical (reserved name)
alpha:  both a local and a global variable. CHECK!!
```

### wrapgrdo.f90

```
Description in plain text:
Single-particle equal-time Green functions are the central object of the code.
The physical single-particle equal-time Green function $G(\tau)$ is updated in wrapgrdo.f90
(down propagation, from $\tau=LTROT$ to $\tau=0$).
The update is sequentially, over all (interacting) lattice sites or lattice bonds.

Input/output variables
in: ntau, integer
inout: gr (ndim,ndim,n_fl), complex
inout: phase, complex
out:

Dependencies
include:
modules: Hamiltonian, MyMats, Hop_mod
interfaces: upgrade
global variables: op_v, phi, nsigma, ndim, n_fl
subroutines: Hop_mod_mmthl, Hop_mod_mmthr_m1, Op_Wrapdo, Upgrade

Things to check:
```

### wrapgrup.f90

```
Description in plain text:
Single-particle equal-time Green functions are the central object of the code.
The physical single-particle equal-time Green function $G(\tau)$ is updated in wrapgrup.f90
(up propagation, from $\tau=0$ to $\tau=LTROT-1$).
The update is sequentially, over all (interacting) lattice sites or lattice bonds.

Input/output variables
in: ntau, integer
inout: gr (ndim,ndim,n_fl), complex
inout: phase, complex
out:

Dependencies
include:
modules: Hamiltonian, Hop_mod
interfaces: upgrade
global variables: op_v, phi, nsigma, ndim, n_fl
subroutines: Hop_mod_mmthr, Hop_mod_mmthl_m1, Op_Wrapup, Upgrade

Things to check:
```

### wrapul.f90

```
Description in plain text:
To stabilize the simulation at the time slice $\tau_{2}=i n_{stab}$,
the Green function has to be recomputed regularly,
based on the stable matrices at an earlier stabilization point, $\tau_{1}=(i-1) n_{stab}$.
These stable matrices result from a singular-value-decomposition of the propagation matrix.
They are computed in wrapul.f90  (down propagation).

Input/output variables
in: ntau1, integer
in: ntau, integer
inout: ulup (ndim,ndim,n_fl), complex
inout: dlup (ndim,n_fl), complex
inout: vlup (ndim,ndim,n_fl), complex
out:

Dependencies
include:
modules: Hamiltonian, Hop_mod, UDV_Wrap_mod
interfaces:
global variables: ndim, n_fl, Op_V, Phi, nsigma,
subroutines:  initd, Op_mmultL, Hop_mod_mmthl, mmult, UDV_Wrap

Things to check:
```

### wrapur.f90

```
Description in plain text:
To stabilize the simulation at the time slice $\tau_{2}=i n_{stab}$,
the Green function has to be recomputed regularly,
based on the stable matrices at an earlier stabilization point, $\tau_{1}=(i-1) n_{stab}$.
These stable matrices result from a singular-value-decomposition of the propagation matrix.
They are computed in wrapur.f90  (up propagation).

Input/output variables
in: ntau1, integer
in: ntau, integer
inout: ur (ndim,ndim,n_fl), complex
inout: dr (ndim,n_fl), complex
inout: vr (ndim,ndim,n_fl), complex
out:

Dependencies
include:
modules: Hamiltonian, Hop_mod, UDV_Wrap_mod
interfaces:
global variables: ndim, n_fl, Op_V, Phi, nsigma,
subroutines:  initd, Op_mmultR, Hop_mod_mmthr, mmult, UDV_Wrap

Things to check:
```

```
Description in plain text:

Input/output variables
in:
inout:
inout:
out:

Dependencies
include:
modules:
interfaces:
global variables:
subroutines:

Things to check:
```

# Module Hamiltonian

*Detailed description of the module Hamiltonian since it will be modified by the users*

The module contains the following subroutines:

### ham_set

It calls the subroutines

- ham_latt
- ham_hop
- ham_v

It reads in the file

- parameters

It sets the variables `ltrot,n_fl,n_sun`. If compiled as a MPI-program, it broadcasts all variables that define the lattice, the model and the simulation process.

### ham_latt

It sets the lattice, by calling the subroutine

- make_lattice(l1_p, l2_p, a1_p, a2_p, latt)

### ham_hop

Setup of the hopping amplitudes between the vertices of the graph (lattice sites and unit cell orbitals). It calls the subroutines

- op_make(op_t(nc,n),ndim
- op_set(op_t(nc,n))

## ham_v

It calls the subroutines

- op_make(op_v(i,nf),1)
- op_set( op_v(nc,nf) )

## s0(n,nt)

It is defined as $s0(n, nt) = 1.d0$. Why? It is superfluous.

## alloc_obs(ltau)

Allocation of equal time and time-resolved quantities.

## init_obs(ltau)

Initializes equal time and time-resolved quantities with zero.

## obser(gr,phase,ntau)

Includes the definition of all equal-time observables (scalars and correlation functions) that are built from the single-particle Green function based on Wick's theorem.

## pr_obs(ltau)

Output (print) of the observables.

## obsert(nt,gt0,g0t,g00,gtt,phase)

Includes the definition of time-resolved observables that are built from the time-resolved single-particle Green function based on Wick's theorem.

# Necessary background information

## Definition of the physical Hamiltonian and its implementation

The physical Hamiltonians that we can simulate have the general form:

$$\mathcal{H} = \sum_{s=1}^{N_{fl}} \sum_{\boldsymbol{x},\boldsymbol{y}} c^{\dagger}_{\boldsymbol{x}s} M_{\boldsymbol{x}\boldsymbol{y}} c_{\boldsymbol{y}s} - \sum_{k=1}^{M} U_k \left[ \sum_{s=1}^{N_{fl}} \sum_{\boldsymbol{x},\boldsymbol{y}} \left( c^{\dagger}_{\boldsymbol{x}s} T^{(k)}_{\boldsymbol{x}\boldsymbol{y}} c_{\boldsymbol{y}s} - \alpha_k \right) \right]^2 . \tag{1}$$

The indices $\boldsymbol{x}, \boldsymbol{y}$ are multi-indices that label sites and spin states: $\boldsymbol{x} = (i, \sigma)$, where $i = 1, \cdots N_{sites}$ and $\sigma = 1, \cdots N_{sun}$, so

$$\sum_{\boldsymbol{x},\boldsymbol{y}} \equiv \sum_{i=1,j=1}^{N_{sites}} \sum_{\sigma=1,\sigma'=1}^{N_{sun}} . \tag{2}$$

Note, that we introduced *two* different labels for the number of spin states (flavours): $N_{fl}$ and $N_{sun}$. The number of correlated sites which is a subset of all sites, is labelled by $M$ ($M \leq N_{sites}$).

I suggest to use a more intuitive notation and to label the hopping matrix by $T$ and the interaction matrix by $V$:

$$\mathcal{H} = \sum_{s=1}^{N_{fl}} \sum_{\boldsymbol{x},\boldsymbol{y}} c^{\dagger}_{\boldsymbol{x}s} T_{\boldsymbol{x}\boldsymbol{y}} c_{\boldsymbol{y}s} - \sum_{k=1}^{M} U_k \left[ \sum_{s=1}^{N_{fl}} \sum_{\boldsymbol{x},\boldsymbol{y}} \left( c^{\dagger}_{\boldsymbol{x}s} V^{(k)}_{\boldsymbol{x}\boldsymbol{y}} c_{\boldsymbol{y}s} - \alpha_k \right) \right]^2 , \tag{3}$$

### Which information does the type *operator* contain?

## Tutorial to set up the Hubbard model

The $SU(2)$ symmetric Hubbard model is given by

$$\mathcal{H} = -t \sum_{\langle i,j \rangle, \sigma} \left( c^{\dagger}_{i,\sigma} c_{j,\sigma} + \text{H.c.} \right) + \frac{U}{2} \sum_{i} \left[ \sum_{\sigma} (c^{\dagger}_{i\sigma} c_{i\sigma}) - 1 \right]^2 . \tag{4}$$

To bring Eq. (4) in the general form (3), we define:

$$\begin{aligned}
N_{fl} &= 1 \\
N_{sun} &= 2 \\
T_{\boldsymbol{xy}} &= -t\delta_{\langle i,j \rangle} \delta_{\sigma,\sigma'} \\
M &= N_{sites} \\
U_k &= -U/2 \\
V^{(k)}_{\boldsymbol{xy}} &= \delta_{i,j} \delta_{i,k} \delta_{\sigma,\sigma'} \\
\alpha_k &= 1/(N_{sites} N_{sun})^2 .
\end{aligned} \tag{5}$$

## Tutorial to set up a lattice

## Installation

### Dependencies

*which software and libraries are needed and which version*

- libraries: LAPACK, BLAS, EISPACK, NAG *They are included in the package, but NAG is not public-domain (?)*
- tools: cmake
- compiler: gfortran or ifort

### Build the GQMC program from source code

*configuration, compile and Installation* In the top level directory, where the README file resides, do:

```
mkdir build
cd build
cmake ..
make
```

## Reference manual

## 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.

## ideas

FAQ, walkthroughs,