QUEST Users' Guide

(version 1.0.0)

Che-Rung (Roger) Lee
Zhaojun Bai
Richard Scalettar

Acknowledgement

Development of QUEST is supported through a SciDAC (Scientific Discovery through Advanced Computing) grant by the U.S. Department of Energy - Office of Science, Advanced Scientific Computing Research and the National Nuclear Security Agency under the contract number DE-FC-02-06ER25793. It is a part of the project on "Modeling Materials at the Petascale: next generation multi-scale quantum simulation software for strongly correlated materials".

Contents

1 Introduction

QUantum Electron Simulation Toolbox (QUEST) is a Fortran 90/95 package that implements the Determinant Quantum Monte Carlo (DQMC) method for quantum electron simulations. The original version of DQMC programs ("legacy codes") have been developed and extensively used for years to study magnetism, metal-insulator transition and superconductivity in the Hubbard model.[1] QUEST, the new version of DQMC simulations, serves three purposes.

1. To improve simulation performance: QUEST has improved the performance of simulations by using new algorithms, like delayed update, and by integrating modern numerical kernels, such as BLAS and LAPACK. A six to eight times speedup had been observed for medium sized simulations.

2. To integrate existing programs: QUEST has integrated many legacy codes by modularizing their computational components, which makes QUEST not only a single program, but a toolbox. The advantages of modularization also include the ease of maintenance and the convenience of program interfacing.

3. To assist new simulations development: QUEST has several properties for developing new simulations, such as the ability of creating new lattice geometries. Several novel simulations had been done by using QUEST.

1.1 Problems that QUEST can solve

QUEST uses a two-dimensional periodic rectangular lattice as the default geometry in the simulation. The lattice size and physical parameters are configurable through an input file. Several physical measurements for the Hubbard model can be calculated by the program, as will be discussed in Section 2.2.

QUEST is stable for interaction strength from $U = 0$ to $U = 16t$ and inverse temperatures $\beta = 0$ to $20/t$. Here U is the onsite repulsion and t is the coefficient of the kinetic energy term, see Section 2.1. However, away from half-filling, $\mu = 0$, error bars are large for large β because of the sign problem.

The current implementation allows hundreds of sites to be simulated on commodity computers within a reasonable time. Appendix A provides performance benchmark for some example problems.

1.2 Organization of the package

The top level files and directories of QUEST include

- `README`: installation guide.
- `make.inc`: configuration file for system dependent parameters.
- `Makefile`: instruction file for make program.

[1] The codes mentioned here were developed by the condensed matter theory group at UCSB including R. L. Sugar, D. J. Scalapino, S. R. White and E. Y. Loh, and maintained by R. Scalettar.

- `SRC`: directory for source code
- `EXAMPLE`: directory for executable programs.
- `DOC`: directory for documentation.

Under `EXAMPLE` directory, several programs are contained.

- `test`: A simple test program for 2 dimensional square lattice.
- `verify`: A program verifying the correctness of QUEST by examining its results against theoretical values of two special cases, $U = 0$ and $t = 0$.
- `tdm`: A test program for time dependent measurements.
- `parallel`: A test program for MPI type parallelization.
- `wrap`: Programs for different input and output formats.
- `sheet`: Programs for multilayer geometry.
- `DCA`: Programs for using DQMC as DCA solver.
- `gemo`: Programs for general geometry.

The details for each program will be illustrated in Appendix A.

1.3 Compilation and execution

The compilation of QUEST can be done through the following steps.

1. Download package from the link

   `http://www.cs.ucdavis.edu/~bai/QUEST/quest1.0.tar.gz`

   and save it to a desired directory.

2. Extract the file by

   ```
   tar -xzf quest1.0.tar.gz
   ```

   which will create a directory `QUEST_1.0`

3. Edit `QUEST\make.inc`. The major changes that should be made include

   (a) `FC`: The Fortran 90 compiler.
   (b) `FC_FLAG`: The flags of Fortran compiler.
   (c) `HOME`: The full path where `QUEST` is installed.
   (d) `BLASLIB` and `LAPACKLIB`: path of BLAS/LAPACK library. The reference implementation is available on

`http://www.netlib.org`

4. Use the command `make` or `make lib` to compile the library. A successful compilation will generate `dqmclib.a` under `QUEST` directory.

5. Go to any example directory, such as `EXAMPLE\test`, and use the command `make` to generate executables. To compile the program in `EXAMPLE\parallel`, one needs `mpifc`, an MPI Fortran compiler, to compile the program.

6. Use `make clean` to clean up the compiled library and programs.

Most executables require input parameters from standard input, the keyboard. A simple way to input those parameters is to use IO redirect. For example, suppose the executable is called `test`. The execution of `test` will be

```
./test < param.in
```

where `param.in` is a file that contains input parameters. In each directory of executables, there will be some sample input files provided. The meaning of each parameter will be explained in the following section.

2 Basic usages

In this section we will introduce the basic input parameters and output formats of QUEST. Those parameters and formats are not universal, since QUEST allows each program to define its own input parameters and output results, as introduced in section 3. However, by walking through those basic parameters and formats, one can understand what QUEST needs and what QUEST can do, and the input/output styles.

2.1 Input file

An input file of QUEST is consisted of a list of parameter assignments,

parameter_name = parameter_value

and arbitrary single line comments, starting with symbol `#`. Each parameter is associated with one of the following types: *integer*, *real*, *real array*, and *string*. The basic parameters include

1. `fname` (string): file name for output files.

2. `n` (integer): total number of sites.

3. `nx` (integer): number of sites in the x-direction.

4. `ny` (integer): number of sites in the y-direction.

5. `nz` (integer): number of sites in the z-direction.

6. `U` (real array): Hubbard interaction parameters.

7. `t` (real array): Hubbard hopping parameters.

8. `mu` (real array): Hubbard chemical potential parameters.
9. `L` (integer): number of time slices.
10. `dtau` (real): discretization parameter.
11. `HSF` (integer): indicator of how Hubbard Stratonovich field is input.
12. `HSFin` (string): file name of input file of HSF.
13. `HSFout` (string): file name of output file of HSF.
14. `nwarm` (integer): number of warmup sweeps.
15. `npass` (integer): number of measurement (sample) sweeps.
16. `nmeas` (integer): frequency of performing equal time measurements.
17. `tausk` (integer): frequency of performing unequal time measurements.
18. `nbin` (integer): number of bins for measurement results.
19. `seed` (integer): random number seed.
20. `north` (integer): frequency of performing orthogonalization in matrix product.
21. `nwrap` (integer): frequency of recomputing Green's function.
22. `difflim` (real): tolerable difference of the matrices computed from different methods.
23. `errrate` (real): tolerable error rate of recomputing.
24. `ntry` (integer): number of sites to be flipped in the global sweep.

These parameters can be roughly divided into three groups:

1. Parameters for Hubbard model:

   Parameters `nx` and `ny` specify the x-dimension and y-dimension of the 2-D rectangular lattice to be simulated. Note that the parameter n, the number of total sites, is equivalent to `nx`×`ny`. Parameter `t`, `mu`, and `U` are used in the Hubbard Hamiltonian

   $$\mathcal{H} = -t \sum_{<i,j>,\sigma} (c^\dagger_{i\sigma} c_{j\sigma} + c^\dagger_{j\sigma} c_{i\sigma}) - \mu \sum_{i,\sigma} n_{i\sigma} + U \sum_i (n_{i\uparrow} - \frac{1}{2})(n_{i\downarrow} - \frac{1}{2}),$$

   for kinetic energy, chemical energy and potential energy respectively. Parameter `L` and `dtau` are for inverse temperature β.

   $$\beta = L\Delta\tau = \frac{1}{T}.$$

   Parameter `HSF` indicates how the Hubbard-Stratonovich Field (HSF) is initialized.

   $$\text{HSF} = \begin{cases} -1, & \text{randomly generating HSF;} \\ 0, & \text{use HSF in the memory;} \\ 1, & \text{read HSF from a file.} \end{cases}$$

If `HSF=1`, then `HSFin` is the input file name. The generated HSF can be also output to a file by specifying the file name in `HSFout`.

2. Parameters for Monte Carlo simulation and physical measurements:

   Parameter `nWarm` and `nPass` in the second group decide how many updates of the entire set of all Hubbard-stratonovich variables will be executed for warm up and measurement sweep. Parameter `nMeas` and `tausk` specify the frequency of performing physical measurements. Parameter `nBin` determines how the computed data is divided into bins. Parameter `ntry` is used to specify how many global flips should be performed per sweep.

3. Parameters regarding numerical concerns.

   Parameter `seed` is used for random number generator. If it is 0, a new seed will be generated from system time. Parameter `nOrth` specifies how often the stabilization algorithm should be performed in the calculation of Green's function. Parameter `nWrap` provides the initial frequency of recomputing Green's function instead of "wrapping" G to the next time slice. In QUEST, `nWrap` will be dynamically adjusted according to the errors of updating. Parameters `difflim` and `errrate` are used for the adjusting algorithm, which specify the tolerable matrix difference and the acceptable error rate.

2.2 Output

The output of QUEST varies in different programs. The goal for this section is to introduce what QUEST can output, and their formats. The output of QUEST can be classified into three types

- Input/configuration parameters.
- Equal time measurements.
- Unequal time measurements.

The input/configuration parameters are as introduced in the previous section. Some of them may be created or changed during the simulation, like `seed` and `nWrap`. Those configuration parameters can help identifying the output results.

In terms of formats, concerned only with measurements, there are three kinds

- Single real number.
- Array of real numbers.
- Array of complex numbers.

If a measurement is a single real number, it will be shown with three terms: name, average, and error. For example,

```
Density :     1.000000 +-     0.000000
```

The measurement formatted in an array of real numbers (or complex numbers) is a *function*, whose arguments can be anything, like the distances between sites. The output of this type enumerates all its function arguments and values. For example, the equal time Green's function of a 4×4 periodic lattice is output like

```
Equal time Green's function:
          dx =  0, dy =  0        0.500000 +-     0.000000
          dx =  1, dy =  0       -0.119358 +-     0.000754
          dx =  2, dy =  0        0.000000 +-     0.000000
          dx =  0, dy =  1       -0.118936 +-     0.000414
          dx =  1, dy =  1        0.000000 +-     0.000000
          dx =  2, dy =  1        0.020050 +-     0.000307
          dx =  0, dy =  2        0.000000 +-     0.000000
          dx =  1, dy =  2        0.019829 +-     0.000184
          dx =  2, dy =  2        0.000000 +-     0.000000
```

The first line is the name of measurements. Below that, the first column is the arguments of the function. The second and the third columns are the averages and errors.

Complex results will be defined similarly with separated error bars for the real part and imaginary part.

For the detailed definition, please refer to the working notes.

2.2.1 Equal time measurements

There are three groups of equal time measurements. The first group is the measurements that aggregate values from entire lattice. The second group is the correlation functions that average pair of sites within the same distance class. The third group measures the pair susceptibilities.

1. Up spin occupancy
2. Down spin occupancy
3. Potential energy
4. Kinetic energy
5. Total energy
6. Density
7. XX Ferromagnetic structure factor
8. ZZ Ferromagnetic structure factor
9. XX Antiferromagnetic structure factor
10. ZZ Antiferromagnetic structure factor
11. Equal time Green's function
12. Density-density correlation function (up-up)
13. Density-density correlation function (up-dn)
14. XX Spin correlation function

15. ZZ Spin correlation function
16. S-wave pair structure factor
17. SX-wave pair structure factor
18. D-wave pair structure factor
19. SXX-wave pair structure factor
20. DXX-wave pair structure factor
21. PX-wave pair structure factor
22. PY-wave pair structure factor
23. PXY-wave pair structure factor
24. PYX-wave pair structure factor

2.2.2 Unequal time measurements

Every unequal time measurement is a function of imaginary time. The format of array typed measurement with two arguments, space and time, is like

```
G(nx,ny,ti)
dx =  0, dy =  0
          0   0.50000 +-   0.00000
          1   0.36076 +-   0.00311
          2   0.27942 +-   0.00328
      ...
dx =  1, dy =  0
          0  -0.12042 +-   0.00162
          1  -0.07183 +-   0.00097
          2  -0.04248 +-   0.00133
      ...
dx =  2, dy =  0
      ...
```

where `dx =  0, dy =  0` are labels of space and followed by a list of time label and corresponding values.

Some measurements are from Fourier transformation, which are complex. Their format will be like

```
 0(   0.05179 +-   0.00270)  +i(   0.00000 +-   0.00000)
 1(   0.05085 +-   0.00248)  +i(  -0.00061 +-   0.00055)
 2(   0.04343 +-   0.00190)  +i(  -0.00067 +-   0.00078)
 3(   0.03755 +-   0.00144)  +i(  -0.00080 +-   0.00051)
 4(   0.03004 +-   0.00099)  +i(  -0.00090 +-   0.00076)
 5(   0.02512 +-   0.00092)  +i(  -0.00100 +-   0.00048)
```

```
6(   0.02152 +-   0.00074)  +i(  -0.00134 +-   0.00031)
7(   0.01829 +-   0.00072)  +i(  -0.00087 +-   0.00055)
8(   0.01499 +-   0.00066)  +i(  -0.00051 +-   0.00055)
9(   0.01332 +-   0.00065)  +i(  -0.00042 +-   0.00035)
```

The numbers inside the first parenthesis are the average and error of the real part and the numbers in the second parenthesis are for imaginary part.

1. Unequal time Green's function: $G(dx, dy, t)$
2. Discrete cosine transformed $G(dx, dy, t)$: $G(qx, qy, t)$
3. Fourier Transformed $G(dx, dy, t)$: $G(dx, dy, w)$
4. Fourier Transformed $G(qx, qy, t)$: $G(qx, qy, w)$
5. Spin-spin correlation function: $\chi(dx, dy, t)$
6. Discrete cosine transformed $\chi(dx, dy, t)$: $\chi(qx, qy, t)$
7. Fourier Transformed $\chi(dx, dy, t)$: $\chi(dx, dy, w)$
8. Fourier Transformed $\chi(qx, qy, t)$: $\chi(qx, qy, w)$
9. S-Wave pair structure factor, vertex and nonvertex
10. SX-Wave pair structure factor, vertex and nonvertex
11. D-Wave pair structure factor, vertex and nonvertex
12. SXX-Wave pair structure factor, vertex and nonvertex
13. DXX-Wave pair structure factor, vertex and nonvertex
14. PX-Wave pair structure factor, vertex and nonvertex
15. PY-Wave pair structure factor, vertex and nonvertex
16. PXY-Wave pair structure factor, vertex and nonvertex
17. PYX-Wave pair structure factor, vertex and nonvertex
18. FTed S-Wave pair structure factor, vertex and nonvertex
19. FTed SX-Wave pair structure factor, vertex and nonvertex
20. FTed D-Wave pair structure factor, vertex and nonvertex
21. FTed SXX-Wave pair structure factor, vertex and nonvertex
22. FTed DXX-Wave pair structure factor, vertex and nonvertex
23. FTed PX-Wave pair structure factor, vertex and nonvertex

24. FTed PY-Wave pair structure factor, vertex and nonvertex

25. FTed PXY-Wave pair structure factor, vertex and nonvertex

26. FTed PYX-Wave pair structure factor, vertex and nonvertex

3 Advanced usages

To assist in creating new simulations,[2] QUEST provides several simple mechanisms to

1. Add new configuration parameters,

2. Create new lattice geometries,

3. Add new measurements.

This section will introduce how to apply them.

3.1 Add new configuration parameters

Section 2.1 lists the basic input parameters, which are enough for current programs. However, for new simulations, additional parameters may be required. QUEST allows new parameters to be specified through current configuration system. For example, in the diluted lattice, in which some sites are randomly removed, the percentage of removal sites can be specified in the input file

```
rmv_ratio = 0.1
```

just like the other parameters.

To add new configuration parameters, the user/developer needs to edit a file called `config.def`. When the subroutine `DQMC_Config_Read` is called, it will first search `config.def` for the definition of parameters. If the file exists, then the program will use the parameter defined in the file. Otherwise, it uses default parameter set, as described in section 2.1.

A parameter in `config.def` is defined by a quintuple:

$$\{\texttt{name, type, isArray, printed, default}\}$$

where

`name:` a string, maximum characters 30, specifying the name of the parameter.

`type:` an integer specifying the data type of the parameter.

$$\begin{cases} \texttt{type} = 1, & \text{real;} \\ \texttt{type} = 2, & \text{integer;} \\ \texttt{type} = 3, & \text{string.} \end{cases}$$

[2] Currently, QUEST is limited to the single band Hubbard model with on site repulsion, but allows very general lattice geometries

`isArray:` a boolean {T,F} specifying whether the parameter is an array. Note, when type is a string, `isArray` cannot be T.

`printed:` a boolean {T,F} specifying whether the parameter will be printed in the output.

`default:` a string, maximum characters 30, specifying the default value of the parameter.[3] QUEST will convert it to a proper data type based on the "`type`"-tuple.

Each tuple is separated by spaces. For example, the parameter `rmv_ratio` can be defined in the `config.def` as

```
rmv_ratio  1 F T 0.1
```

The file `config.def` should be placed in the directory where the executable is.

3.2 Create new lattice geometries

In QUEST, geometry dependent variables are placed together in a derived data type `Struct`. To create a new lattice geometry, one needs to fill out the data fields in `Struct`. There are several ways to achieve this; each has its advantages and drawbacks.

1. Write a program to fill out the data fields: most efficient way in execution, but need to write programs for different geometries.

2. Use primitive cell definition as input:[4] Most compact representation of general geometry, but requires understanding the definitions of primitive cell.

3. Input data for each data field from files: Most easy way to create a new lattice geometry, but less flexible and efficient.

In this section, we will illustrate the data fields in `Struct` and the file formats for method 3. Note not all the data fields are essential for simulations. Some of them are just for a particular measurements. When lacking one or some of the fields, QUEST will skip the corresponding measurements.

3.2.1 `Struct`

Derived data type `Struct` is defined as

```
type Struct
   integer  :: nSite               ! number of sites
   character(gname_len):: name     ! name of the structure
   integer, pointer    :: dim(:)   ! dim of the geometry

   integer  :: n_t                 ! number of hopping types
   type(CCS):: T                   ! hopping matrix
```

[3] Note, even a parameter is an array, its default value can be only set by one number.
[4] This feature is still under development.

```
   integer  :: n_b                ! number of neighbors types
   type(CCS):: N                  ! neighborhood matrix

   integer  :: nClass             ! number of distance classes.
   integer, pointer  :: D(:,:)    ! distance classes
   integer, pointer  :: F(:)      ! counts for each dist class.
   character(label_len), pointer :: &
                         label(:) ! label for distant class.
   integer  :: nGroup             ! number of diff singletons.
   integer, pointer  :: map(:)    ! site classification

   real(wp), pointer :: W(:,:)    ! wave functions
   integer  :: nWave
   character(label_len), pointer :: &
                          wlabel(:)! label for wave functions.
   real(wp), pointer :: P(:)      ! phase assignment

   real(wp), pointer :: FT(:,:)   ! Fourier Transform matrix

   logical::checklist(N_CHECKLIST) ! flags
end type Struct
```

Here are the detailed comments for each field.

- Sites must be numbered from 1, with continuous numbering.
- String `name` is of length 80 characters, used in display.
- Vector `dim` is used to hold dimension parameters. For example, for a two dimensional square lattice, `dim=(nx,ny)`, where `nx` and `ny` are the number of sites in x direction and in y direction. This field can be of arbitrary length. QUEST does not use this field directly.
- Hopping matrix `T` stores the indices of hopping parameter `t` for adjacent sites.[5] It is normally a sparse matrix; and therefore is represented in the Compressed Column Storage (CCS) format.[6] Field `n_t` is the number of different hopping parameters.
- Neighboring matrix `N` is similar to `T`, and is also represented in the CCS format. It is used in pair measurements, for which the link indices should be consistent with the wave function `W`. Field `n_b` is the number of different links.
- The distance classification is represented by `D`, `F` and `label`. Two pairs of sites are in the same class if they have equivalent separation in geometry in question. The number of classes is specified by `nClass`. The class index is also started from 1. The number of

[5] Two sites i, j are called adjacent if electrons can hop from site i to site j.
[6] The detail of CCS format can be found in `http://www.netlib.org`.

pairs in each class is stored in `F`. String array `label` gives a label for each class, used in output. Matrix `D` records the classification for each pair of site. Number in `D(i,j)` denotes the class index for site i and j.

- The vector `map` classifies sites. Site i and site j are in the same class if they have the same physical parameters, like U and μ. The number `nGroup` denotes the number of site classes.
- Matrix `W` defines wave functions for the pair measurements, which are related to various spherical harmonic functions. The number of functions is specified by `nWave`. Each function has `n_b` elements, and is stored in a column of `W` matrix.
- The phase assignment vector `P` gives each site $\{+1, -1\}$ so that adjacent sites have opposite phases. This is used in spin correlation measurements, and in Green's function calculation when $\mu = 0$. This is useful only for bipartite lattices.
- Matrix `FT` is the Fourier transformation matrix for distance classes, which is used in time dependent measurements.
- Vector `checklist` is a set of flags that indicates which data fields are assigned. The flags include

```
STRUCT_INIT   = 1
STRUCT_DIM    = 2
STRUCT_ADJ    = 3
STRUCT_CLASS  = 4
STRUCT_WAVE   = 5
STRUCT_NEIG   = 6
STRUCT_PHASE  = 7
STRUCT_FT     = 8
```

3.2.2 Files for geometry definition

The file name of geometry definition is specified in the configuration file with parameter name `gfile`. For example,

```
gfile = strip.def
```

tells QUEST to use the file `strip.def` as geometry definition. The format of the geometry definition is similar to the configuration file.

1. The symbol `#` is used to start a comment.
2. The data fields are set by the assignment

   ```
   data_field_name = data_field_value
   ```

3. Some fields depend on the other. Depended fields should be declared first.

T → nSite
B → nSite
D → nSite
FT → nClass
W → n_b and nWave

4. Not every field in `Struct` need be defined in the file. For example, vector `F` and `map` will be derived from matrix `D`.

5. Array type assignment, including matrix or vector, is different from that in a configuration file. The right-hand-side of assignment should be the number of lines to read. For example

   ```
   D = 64
   ```

   means there are 64 immediate lines to read for matrix `D`. Those lines are called *content lines*. Note NO empty lines are allowed between content lines.

6. The format for a content line of a matrix is

   ```
   i j value
   ```

   where `i` is the row index, `j` is the column index, and `value` is the (`i,j`) element of the matrix.

7. Format for a content line of a vector is

   ```
   value
   ```

   Those values should be ordered sequentially, since indices are assumed to be implicitly embedded.

An example of how to create new geometry using method 3 can be found in the example program in `EXAMPLE\gemo`.

3.3 Add new measurements

New measurements need be made through programming in QUEST. Several subroutines can be used to create new measurements. The standard procedure to add a new measurement includes three steps.

1. Measuring.

2. Binning.

3. Postprocessing. (Statistics, Fourier transformation, output.)

3.3.1 Measuring

Several components may be needed to create a new measurements.

1. **Equal time Green's function**: Equal time Green's function is defined in the module `dqmc_gfun`, which will be initialized automatically when the subroutine `DQMC_Hub_Init` is called. Suppose `Hub` is typed `Hubbard`, the major data type of entire simulation. The Green's function matrices, spin-up and spin-down, can be obtained from

   ```
   Hub%G_up%G
   Hub%G_dn%G
   ```

   And the signs of their determinants are recorded in

   ```
   Hub%G_up%sgn
   Hub%G_dn%sgn
   ```

2. **Unequal time Green's function**: Unequal time Green's function, denoted G^{τ}_{ρ}, $\rho \in \{\uparrow, \downarrow\}$, is defined in the module `dqmc_gtau`. Unlike equal time Green's functions, G^{τ} are not essential in the simulation. Therefore, user needs to initialize it by calling `DQMC_Gtau_Init` explicitly. The construction of G^{τ} can be made in two ways. The first way is to call `DQMC_Gtau_Big`, which returns entire G^{τ}_{ρ}. The second method is to invoke `DQMC_MakeGtau`, which returns block submatrices of G^{τ}. Since the signs of unequal time Green's functions are the same as the equal times, one can obtain the signs from `Hub%G_up%sgn, Hub%G_dn%sgn`.

3. **Parameters of Hamiltonian**: Parameters, such as t, μ and U, are stored in the data type `Hubbard`. The access is straightforward.

4. **Geometry related information**: Geometry information, such as hopping matrix, can be obtained from the data type `struct`, which is introduced in section 3.2.

3.3.2 Binning

In order to generate error bars, measurements in QUEST need be grouped into bins. Currently, equal divided binning strategy is used, which means the measurements are evenly divided by the total number of bins. Measurements in the same bin are averaged. The total number of bins are stored in the variable `nbin`.

3.3.3 Statistics

There are two special properties for some of the physical measurements produced by the DQMC method.

1. The distribution is not normal.

2. Measurements are weighted with signs of the determinants of Green's functions, for which the average needs be normalized by the average of signs.

QUEST uses *Jackknife* resampling technique in error estimation. Two subroutines are provided to perform the statistics: `DQMC_JackKnife` and `DQMC_SignJackKnife`. The former is for error estimation of signs; the latter is used for other measurements. Those subroutines are defined in module `dqmc_util`.

3.3.4 Fourier transformation

For unequal time measurements, there are two possible Fourier transformations to be applied: transformation on the real space and transformation on the time domain. For the transformation on real space, since it is geometry dependent, a transformation matrix `FT`, defined in `Struct`, is required.[7] Once the matrix is available, the transformation is just a matrix-matrix multiplication. In the module `dqmc_tdm`, QUEST provides a subroutine `DQMC_DCT_Space` for the space transformation.

The Fourier transformation on the time domain is an integration. The numerical procedure is

1. Refine the time grid.
2. Interpolate the refined data points.
3. Integrate on the interpolated data with Fourier coefficients.

In step 1, QUEST evenly subdivides the time domain by the given parameter `nitvl`. In step 2, QUEST uses spline interpolation, which is supported by the subroutine `DQMC_Spline`. Step 3 requires an additional Fourier matrix, which can be generated from the subroutine `DQMC_Make_FTM`. The entire procedure is coded in the subroutine `DQMC_FT_Time`.

3.3.5 Printing

QUEST has two subroutines that print out arrays of numbers. Subroutine `DQMC_Print_RealArry` prints out an array of real numbers; `DQMC_Print_ComplexArray` prints out an array of complex numbers. The title of the measurements and the labels of each array items are required for those two functions.

A Example programs

Eight example programs are included in the `EXAMPLE` directory. Here is a short introduction for each of them.

A.1 `test`

Program `test` demonstrates the simplest usage of the QUEST. Besides the timing commands, there is only one line in the program, which runs DQMC simulation on a two dimensional periodic rectangular lattice. In spite of its simplicity, this program can be configured dynamically for different lattice size, Hubbard parameters and execution iterations. Four sample configurations are included within this program, as showing below.

[7] see section 3.2 for more details.

Configuration	Lattice size	Time slice	Running time
`small.in`	4×4	12	2 second
`medium.in`	8×8	48	198 second
`large.in`	16×16	96	15,629 second

Their execution time is also presented. This result is obtained by using checkerboard method, 1000 warmup steps and 5000 measurement steps, with Intel MKL BLAS and LAPACK libraries, on an Intel Core 2 Duo 2.4GHz processors (but only run in one core).

A.2 `verify`

The `verify` program examines the correctness of the execution results of two special cases, single site ($t = 0$) and no Coulomb interaction ($U = 0$). Each test case runs through 9 configurations.

The correctness of those results are checked against the theoretical results. Statistically, 63.2 percent of the computed results are expected to have errors smaller than one standard error, and 86.5 percent of results should be within two standard errors. The verification program runs 1000 warm-up sweep and 5000 measurement sweep for $t = 0$ cases, which gives error bars of 0.33% on the energy and less than 0.22% on the spin correlation `SpinXX`.

A.3 `tdm`

This example program demonstrates how unequal time measurements can be performed. This program also shows the flexibility of using the subroutines in QUEST. Four sample configurations, the same those in `test`, are available for this program. The time dependent measurements are made every 10 measurement sweeps. Their execution time is summarized in the following table.

Configuration	Lattice size	Time slice	Running time
`small.in`	4×4	12	3 second
`medium.in`	8×8	48	266 second
`large.in`	16×16	96	19446.18 second

A.4 Others

- `parallel`: This program uses MPI to parallelize the measurement steps. The compilation of this program requires `mpif90`. The base compiler of `mpif90` should be the same as the one used in compiling library.

- `wrap`: This directory contains a module `dqmc_wrapper` and a program `tw`. The module wraps all computation components of QUEST into several simpler functions. The program `tw`, test wrapper, can perform equal time and unequal time measurements.

- `sheet`: This directory contains a module and programs for the multilayer lattice. Module `dqmc_sheet` defines the multilayer geometry. Program `meas1` and `meas2` provides new measurements for the multilayer structure.

- `geom`: The program inside this directory tests the general geometry method 3 mentioned in section 3.2.
- `DCA`: This is a project that interfacing QUEST with other programs. Basically, the entire project uses DQMC as a DCA solver. This demonstrates how to use QUEST as a library in other programs.