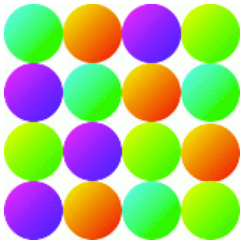
QUEST User's Manual

April, 2012



Contents

1 Introduction

QUantum **E**lectron **S**imulation **T**oolbox (QUEST) is a FORTRAN 90/95 package for performing determinant quantum Monte Carlo (DQMC) simulations for strongly correlated electron systems. Its development was motivated by a FORTRAN 77 DQMC code[1] written and maintained by Richard Scalettar at the University of California, Davis (UCD). Currently the development of QUEST is led by Zhaojun Bai and Richard Scalettar at UCD. Several research groups have also participated the effort: Mark Jarell (Louisiana State University), Eduardo D'Azevedo and Thomas Maier (ORNL), Sergey Savrasov (UCD), and Karen Tomko (Ohio Supercomputer Center).

In QUEST 1.0, structure of the legacy code has been modernized. In particular, BLAS and LAPACK numerical linear algebra libraries and new techniques such as delayed-updating are integrated into QUEST's computational kernel. These developments have improved the efficiency of DQMC simulation and generated fruitful research applications. Recognizing the increasing appeal of using hybrid multicore processors and graphics processing unit (GPU) systems, QUEST 2.0 is released with a new optimized computational kernel designed for such heterogeneous multicore CPU + GPU architecture.

2 Building QUEST

2.1 Basic install

QUEST is a collection of FORTRAN libraries that performs auxiliary-field type quantum Monte Carlo simulations. In order to build QUEST, you will need a FORTRAN 90/95 compiler and BLAS/LAPACK libraries.

- We recommend the latest FORTRAN compiler for better compatibility. QUEST has been tested with GNU's `gfortran`, however, Intel's `ifort` or Portland Group's `pgf90` should be able to compile QUEST too.
- Because QUEST relies heavily on BLAS and LAPACK to perform numerical linear algebra operations, an open source BLAS/LAPACK package has been included in the package. QUEST is also compatible with commercial BLAS/LAPACK libraries such as Intel's Math Kernel Library (MKL) or AMD's AMD Core Math Library (ACML).
- GPU support is through CUDA/CUBLAS parallel programming environment. Information on building CUDA/CUBLAS development tools can be found from CUDA's website `http://developer.nvidia.com/cuda-downloads`.
- QUEST also requires `make` to build its libraries and applications. Most LINUX machine should have GNU make. In case you don't, GNU make can be obtained from the GNU website `http://www.gnu.org/s/make/`.

The latest stable snapshot of QUEST can be downloaded from:

`http://quest.ucdavis.edu/download.html`

[1] Known as the legacy code.

After the tarball file is unzipped, a directory `/QUEST` is created which contains the following files and subfolders:

- `make.inc`: System configuration parameters.
- `Makefile`: GNU `make` instructions. Do not modify this file unless you know what you are doing.
- `/SRC`: QUEST source codes.
- `/EXAMPLE`: Applications and driver routines.
- `/BLAS`: BLAS library.
- `/LAPACK`: LAPACK library.

QUEST needs to gather system environment information for GNU make. This is done through the configuration file `make.inc`. In particular, you need to specify the following variables:

- `DQMCLIB`: Path where the compiled QUEST library `libdqmc.a` will reside. Default is `$(HOME)/libdqmc.a`.
- `FC`: Name of the FORTRAN 90/95 compiler. Default is `gfortran`.
- `FC_FLAG`: Compiler flags. For example: optimization level, debugging flag, ... etc.
- `CXX`: C++ compiler. Default is `gcc`.
- `CXX_FLAG`: Optional C++ compiler flags.
- `LAPACKLIB`: Linking interface for BLAS and LAPACK respectively.
- `CXXLIB`: C++ library linking interface.
- `CUDAPATH`: Path to the CUDA/CUBLAS library.
- `LIB`: Top level library linking interface. Default is `$(LAPACKLIB) $(CXXLIB) $(CUDALIB)`. One can include other external libraries here if necessary.

Modify `make.inc` to suit your computing environment. Once you have set up `make.inc`, QUEST can be compiled by launching the following shell commands:

```
$> make
```

If everything goes well, the compilation will generate `libdqmc.a` in the QUEST package home directory. The library `libdqmc.a` contains all necessary components for DQMC simulations.

2.2 Multicore processor and GPU support

The installation procedure described in the previous section enables QUEST to run on a single processor. Support to multicore processors and GPU is included in the latest version of QUEST. They can be enabled through compiler flags which we briefly describes here.

- Multicore processors

  The major portion of CPU time in a DQMC simulation is used in computing the Green's function G. The evaluation of G involves matrix multiplication and inversion. These operations are done by calling numerical linear algebra libraries BLAS/LAPACK. The BLAS/LAPACK library included in the QUEST package is not optimized and can only run on a single processor. QUEST can take advantage of multicore processors and improve its performance by linking multithreaded BLAS/LAPACK such as Intel's Math Kernel Library (MKL) or AMD's AMD Core Math Library (ACML). Please consult the documentation of MKL or ACML for details of setting up multithreaded environment if any of these libraries is available in the user's system. In addition, the `BLASLAPACK` variable in the `make.inc` needs to be edited accordingly.

- GPU support

  To further boost performance, QUEST now contains experimental support for GPU accelerators using the Nvidia CUDA development tools. By default, the GPU support is not actived. To enable the GPU support, please uncomment all the CUDA related variables in `make.inc` and activate the flag `DQMC_CUDA` in `PRG_FLAGS`.

2.3 Verify installation

To verify the installation, go to the directory `./EXAMPLES/verify` and launch the following commands

```
$> cd ./EXAMPLES/verify
$> make
$> ./verify
```

This will compile and run the code `verify`. This little program computes observables for the non-interacting and single-site Hubbard model, and compares the results with exact solutions. If the numerical and exact results are in agreement, then the installation is complete and QUEST is now ready to be used. The subfolder `/EXAMPLES` has several examples and driver routines that demonstrate how `libdqmc.a` is implemented in real applications. We will give a simple example in the next section.

3 Example: 2D Hubbard model on a square lattice

This section presents a basic implementation of QUEST and describes a minimum number of input parameters required for configuring a DQMC simulation. The example is the 2D one-band Hubbard model on a square lattice. The simulation code can be found under `./EXAMPLES/test`. Enter the directory and execute the shell command

```
$> cd ./EXAMPLES/test
$> make
```

will generate an executable `test`. A sample input file `sample.in` for a 4×4 square lattice with $U = 4t$ at half-filling $\langle n \rangle = 1$ and temperature $\beta = 8t$ is provided. To run the simulation, simply execute the command:

```
$> ./test < sample.in > sample.out
```

This will launch the simulation and redirect all standard outputs to the file `sample.out`, and that's it! Now let's take a closer look at the input file.

3.1 Input file

A QUEST input file is a text file containing a collection of keywords and their values. Broadly speaking, QUEST keywords fall into two categories: keywords that set up the Hubbard Hamiltonian, and those control the dynamics of Monte Carlo simulation. Keyword value is specified by a single line statement:

```
keyword = value
```

QUEST will bypass any line that begins with the pound sign `#`. In the present example, the sample input file has the following keywords

- `nx` (integer): linear dimension of the lattice in the x-direction.
- `ny` (integer): linear dimension of the lattice in the y-direction.
- `U` (real): strength of the onsite interaction U.
- `t_up` (real): hopping integral for spin-$\uparrow$ electrons.
- `t_dn` (real): hopping integral for spin-$\downarrow$ electrons.
- `mu_up` (real): chemical potential $\mu_\uparrow$ for spin-$\uparrow$ electrons.
- `mu_dn` (real): chemical potential $\mu_\uparrow$ for spin-$\downarrow$ electrons.
- `L` (integer): number of imaginary time slices L. In the DQMC simulation, the inverse temperature β is discretized into L "imaginary time" intervals $\beta = L\Delta\tau$.
- `dtau` (real): size of the imaginary time (Trotter) step $\Delta\tau$.
- `nwarm` (integer): number of sweeps in the warmup (thermalization) phase.
- `npass` (integer): number of sweeps in the measurement phase.
- `nbin` (integer): determines the number of bins for measurement data. In the current example, physical observables are measured at each Monte Carlo sweep during the measurement phase. In order to reduce autocorrelation, QUEST groups measurement data into several bins before analyzing statistics.
- `seed` (integer): random number seed. If `seed` is set to zero, QUEST will generate a seed automatically.

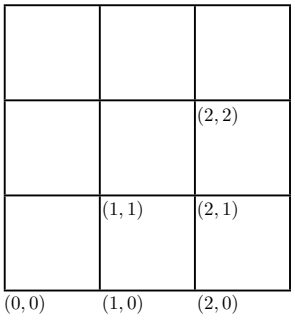
The above keywords constitute a minimal set of parameters to perform DQMC simulation of the 2D Hubbard model.

3.2 Outputs

QUEST is built to perform three types of measurement for the Hubbard model

- Energy : total energy, kinetic and potential energy, specific heat, ... etc.
- Equal-time measurement : one-particle Green's function, correlation functions, static structure factors, local one-body quantities such as density, ... etc.
- Imaginary-time dependent measurement : dynamic structure factor, susceptibility, spectral function, ... etc. Some of these measurements require numerical analytic continuation routines such as MaxEnt.

Depending on the problem, one can incorporate one or several of these measurements into the program by calling appropriate subroutines. Here we briefly describe the top-level QUEST simulation report. Some features of the report are shared with other measurement results.



Simulation results for the 2D Hubbard model on a 4×4 lattice are quoted below. The first section of the report lists input parameters and information showing the behavior of the Monte Carlo run. The second section contains the total average sign of fermion determinant as well as its spin-$\uparrow$ and spin-$\downarrow$ components. In this and the following parts, measurement data is reported as the combination

`average` $\pm$ `error`

where the first number is the average value and the second number is the corresponding statistical error. The next section summarizes some of the equal-time observables such as average density, energy, double occupation, and specific heat, ... etc. The remaining part of the report contains several correlation function results. In these printouts, the first two columns represent the x and y components of the relative distance vector $\boldsymbol{r}$ for the correlation function $C(\boldsymbol{r})$. Because of the symmetry of the 2D square lattice, there is only 6 inequivalent distance vectors on a 4×4 lattice, as indicated by the figure on the left.

4×4 Hubbard model simulation report:

```
 2D Periodic Rectangular Lattice (Nx=  4, Ny=  4) total sites=   16
                           U :            4.000000
                        t_up :            1.000000
                        t_dn :            1.000000
                       mu_up :            0.000000
                       mu_dn :            0.000000
              Time slice (L) :           80
             Number of sites :           16
                        dtau :            0.100000
               beta (dtau*L) :            8.000000
     Number of warmup sweep :          100
Number of measurement sweep :          500
   Frequency of measurement :            2
                Random seed :        32198
 Frequency of recomputing G :           17
Global move number of sites :            0
                Accept count :       364727
```

```
              Reject count :       340652
   Approximate accept rate :            0.517065
                     gamma :            0.400003
         Type of matrix B : Dense MatB
        Type of matrix HSF : 0/1
================================================================================
 Sign of equal time measurements:
                  Avg sign :   0.10000000E+01 +-  0.00000000E+00
                Avg up sign :   0.57415074E+00 +-  0.12719889E+00
                Avg dn sign :   0.57415074E+00 +-  0.12719889E+00
================================================================================
 Equal time measurements:
         Up spin occupancy :   0.49523679E+00 +-  0.79708933E-02
       Down spin occupancy :   0.50476321E+00 +-  0.79708933E-02
            <U*N_up*N_dn>  :   0.47648778E+00 +-  0.32664099E-02
            Kinetic energy :  -0.13169170E+01 +-  0.49848571E-02
              Total Energy :  -0.84042918E+00 +-  0.39851362E-02
                   Density :   0.10000000E+01 +-  0.00000000E+00
               Chi_thermal :  -0.20480000E+04 +-  0.20463631E-11
             Specific heat :   0.75402294E+04 +-  0.50296593E+02
  XX Ferro structure factor :   0.34122407E-01 +-  0.95925025E-01
  ZZ Ferro structure factor :   0.79273784E-01 +-  0.21988555E-01
     XX AF structure factor :   0.31267580E+01 +-  0.35735166E+00
  Root Mean Square of XX AF :   0.69721543E+01 +-  0.19900130E+01
     ZZ AF structure factor :   0.39603736E+01 +-  0.47743189E+00
  Root Mean Square of ZZ AF :   0.47317626E+01 +-  0.47853923E+00
================================================================================
 Mean Equal time Green's function:
  0  0                         0.50000000E+00 +-  0.00000000E+00
  1  0                        -0.16461462E+00 +-  0.62310714E-03
  2  0                         0.21962868E-16 +-  0.19949555E-17
  1  1                        -0.15155052E-15 +-  0.11403475E-17
  2  1                         0.47507936E-01 +-  0.37867763E-03
  2  2                        -0.61854286E-18 +-  0.26400170E-17
================================================================================
 Up Equal time Green's function:
  0  0                         0.50476321E+00 +-  0.79708933E-02
  1  0                        -0.16461462E+00 +-  0.62310714E-03
  2  0                        -0.23313943E-02 +-  0.26568535E-02
  1  1                        -0.60700479E-03 +-  0.22182014E-02
  2  1                         0.47507936E-01 +-  0.37867763E-03
  2  2                         0.39101815E-02 +-  0.59211124E-02
================================================================================
 Down Equal time Green's function:
  0  0                         0.49523679E+00 +-  0.79708933E-02
  1  0                        -0.16461462E+00 +-  0.62310714E-03
  2  0                         0.23313943E-02 +-  0.26568535E-02
  1  1                         0.60700479E-03 +-  0.22182014E-02
  2  1                         0.47507936E-01 +-  0.37867763E-03
  2  2                        -0.39101815E-02 +-  0.59211124E-02
================================================================================
 Density-density correlation fn: (up-up)
  0  0                         0.50000000E+00 +-  0.00000000E+00
  1  0                         0.16687791E+00 +-  0.74883696E-02
  2  0                         0.29504064E+00 +-  0.84888569E-02
  1  1                         0.29327023E+00 +-  0.85469246E-02
  2  1                         0.19944863E+00 +-  0.79376430E-02
  2  2                         0.29155366E+00 +-  0.68663828E-02
================================================================================
 Density-density correlation fn: (up-dn)
  0  0                         0.11912194E+00 +-  0.81660248E-03
  1  0                         0.31057397E+00 +-  0.74355737E-02
  2  0                         0.20225761E+00 +-  0.85132114E-02
  1  1                         0.20425278E+00 +-  0.85687879E-02
  2  1                         0.29832131E+00 +-  0.79459790E-02
```

```
  2  2                          0.20415573E+00 +-  0.68584300E-02
================================================================================
 XX Spin correlation function:
  0  0                          0.76175611E+00 +-  0.16332050E-02
  1  0                         -0.26332216E+00 +-  0.37806808E-01
  2  0                          0.14099770E+00 +-  0.29854957E-01
  1  1                          0.10558928E+00 +-  0.18005604E-01
  2  1                         -0.12325729E+00 +-  0.17868179E-01
  2  2                          0.11433156E+00 +-  0.17090944E-01
================================================================================
 ZZ Spin correlation function:
  0  0                          0.76175611E+00 +-  0.16332050E-02
  1  0                         -0.28739212E+00 +-  0.29847062E-01
  2  0                          0.18556607E+00 +-  0.34003672E-01
  1  1                          0.17803490E+00 +-  0.34230725E-01
  2  1                         -0.19774536E+00 +-  0.31767055E-01
  2  2                          0.17479586E+00 +-  0.27448367E-01
================================================================================
 Average Spin correlation function:
  0  0                          0.76175611E+00 +-  0.16332050E-02
  1  0                         -0.27134548E+00 +-  0.21647718E-01
  2  0                          0.15585382E+00 +-  0.15909091E-01
  1  1                          0.12973782E+00 +-  0.57140133E-02
  2  1                         -0.14808665E+00 +-  0.89007566E-02
  2  2                          0.13448633E+00 +-  0.88719643E-02
================================================================================
 Pairing correlation function:
  0  0                          0.59560972E-01 +-  0.40830124E-03
  1  0                          0.11274059E-01 +-  0.61437258E-04
  2  0                         -0.13508751E-02 +-  0.46068373E-04
  1  1                         -0.12384929E-02 +-  0.55802886E-04
  2  1                          0.11150343E-02 +-  0.27683035E-04
  2  2                         -0.21453031E-02 +-  0.65845693E-04
================================================================================
 Pair measurement:
           S-Wave :             0.19863264E+00 +-  0.11543101E-02
          SX-Wave :             0.58009382E+00 +-  0.24526044E-02
           D-Wave :             0.54811387E+00 +-  0.27145479E-01
         SXX-Wave :             0.22679804E+00 +-  0.76105828E-02
         DXX-Wave :             0.54727617E-01 +-  0.15390399E-01
          PX-Wave :             0.15175323E+00 +-  0.12115481E-01
          PY-Wave :             0.14888977E+00 +-  0.12146828E-01
         PXY-Wave :             0.21852935E+00 +-  0.44857653E-02
         PYX-Wave :             0.22042521E+00 +-  0.45145632E-02
================================================================================
```

In addition to these variables, advanced parameters are also available for a more in-depth control of the Monte Carlo simulation. For example:

- Frequency of recomputing the equal-time Green's function from scratch.
- Frequency of matrix reorthogonalization.
- Methods of stabilizing matrix products.
- Global moves.
- Methods of initializing the Hubbard-Stratonovich fields.

These advanced control variables will be initialized automatically if they are not given in the input file. The values are safe to use for typical simulations with U and β less than the bandwidth.

4 Lattice geometry

Variables describing lattice properties are grouped together in a derived data type `Struct`. It contains a collection of data fields responsible for building a lattice. These include lattice dimension, geometry, hoppings, inequivalent distance class, pair-field operator form factors, ... etc. To create a new type of lattice, it is necessary to construct the corresponding `Struct`. The present version of QUEST has three ways of building a lattice:

- Define `Struct` directly in a driver routine.
- Import from an external file containing explicit geometry information (free form import).
- Build the lattice using primitive cell and super lattice (general geometry).

We will discuss the last two methods later in this section. Before that, let us walk though the definition of the data type `Struct`.

```
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

   integer  :: n_b                            ! number of neighbors types
   type(CCS):: B                              ! 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
```

There are total 19 data fields whose meanings are described as follows.

1. `name`: A string that stores the name of the geometry structure. Maximum length is 80 characters.
2. `dim`: An integer vector containing dimension information of the lattice. For example, a two dimensional square lattice has `dim(2)=(nx,ny)`, where `nx` and `ny` are the number of sites in x- and y-direction respectively. This field can be of arbitrary length. QUEST does not use this field directly.
3. `n_t`: Denotes the number of different hoppings on the lattice.

4. `T`: Stores the indices of hopping parameter `t` for adjacent sites.[2] It is normally a sparse matrix; and therefore is represented in the Compressed Column Storage (CCS) format.[3]

5. `n_b`: Specifies the number of form factor terms in pair-field operators. See the definition of `W` below for more details.

6. `B`: Stores the indices of pairing links. The link indices should be consistent with the form factor array `W`. Similar to `T`, and the array `B` is also stored in the CCS format.

7. Because of symmetry, lattice sites can be grouped together by point group operations. As a result, relative distance between any pair of sites falls into several inequivalent distance classes. The number of such classes is denoted by `nClass`.

   Within each distance class, there are many pairs of sites that have the same relative distance. This number is stored in the array `F`. The string array `label` stores the label for each class, and will be used in QUEST simulation outputs.

   Array `D` Records the classification for each pair of sites. `D(i,j)` denotes the class index for site i and j.

8. `map`: A vector that classifies sites. Site i and site j are in the same class if they have the same physical parameters, like U and μ.

9. `nGroup`: Denotes the number of site classes.

10. `W`: An array that contains the form factors of pair-field operators. The number of desired pairing measurements is specified by `nWave`. Each pair-field operator has `n_b` elements, and is stored in a column of the array `W`.

11. 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 at half-filling $\mu = 0$.

12. `FT`: An array of Fourier transformation matrix for distance classes. `FT` is required in time-dependent measurements.

13. `checklist`: A vector that has a set of flags indicating 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
```

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

4.1 Define `Struct` using free form import

The geometry definition can be imported through the flag `gfile` of the input file. For example, the statement

```
gfile = square.def
```

asks QUEST to read in the file `square.def` for lattice definition. The geometry definition uses free form input and is essentially identical in format to the input file described in Sec. 3.1. Any line begins with a pound sign `#` is treated as a comment. In a `.def` file, each data field is set by the statement

```
data_field_name = data_field_value
```

There are few aspects of `.def` file that we like to address.

1. Allocatable arrays `T, B, D, FT` and `W` requires dimensional information:

   $$\begin{aligned} \texttt{T} &\rightarrow \texttt{nSite} \\ \texttt{B} &\rightarrow \texttt{nSite} \\ \texttt{D} &\rightarrow \texttt{nSite} \\ \texttt{FT} &\rightarrow \texttt{nClass} \\ \texttt{W} &\rightarrow \texttt{n\_b} \text{ and } \texttt{nWave} \end{aligned}$$

   Therefore `nSite, nClass, n_b, nWave` should be declared before array assignment.

2. Vector `F` and `map` will be derived from array `D` when they are not given in the `.def` file.

3. Array assignment (matrix or vector) should always begin with specifying the number of elements. For example,

   ```
   D = 64
   ```

   means that there are 64 elements in array `D`. This will trigger QUEST to read in 64 consecutive lines immediately. These lines are called *content lines*. **NO** empty lines are allowed between content lines.

   - Matrix content line format:

     ```
     i j value
     ```

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

   - Vector content line format:

     ```
     value
     ```

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

An example of free form geometry definition for a 4×4 square lattice is provided in Appendix A.

4.2 Define `Struct` using general geometry

This section focuses on generating a lattice using primitive cell vectors, also known as general geometry. The general geometry input file has the extension `.geom`. Like free form import discussed in the last section, `.geom` file is imported via the top-level input flag `gfile`. For instance, the statement

```
guile = square.geom
```

will read in primitive cell information from the file `square.geom`.

There are 11 flags that may be specified in a `.geom` file. Not all of these flags are required, but when they are given they must be specified exactly as shown below.

1. `#NDIM`: An integer. The number of dimensions in your system (1, 2, or 3 only). The value of this quantity will change the dimension of the input for `#SUPER`, `#K-POINT`, and `#PHASE`.

2. `#PRIM`: A 3×3 real array. The primitive vectors that define the basis for the lattice in cartesian coordinates. The format is

   ```
   a1x   a1y   a1z
   a2x   a2y   a2z
   a3x   a3y   a3z
   ```

   For a two dimensional square lattice, the primitive vectors would be

   ```
   1.0   0.0   0.0
   0.0   1.0   0.0
   0.0   0.0   1.0
   ```

3. `#SUPER`: An `NDIM`$\times$`NDIM` integer array. Defines the supercell of your lattice. The coeffiecients are integers that weight against the primitive vectors. For a `NDIM` $= 2$ system, we have

   ```
   S1x   S1y
   S2x   S2y
   ```

4. `#K-POINT`: An `NDIM` real vector. Defines the original k-point. At the moment, the only supported value for the k-point is the vector $(0, 0, 0)$. Eventually, this will be modified to allow for choosing amongst all k-points.

5. `#ORB`: A listing of the atomic sites in the primitive cell. The format for a site is

   ```
   label   x   y   z   #N
   ```

   where `label` is a string, `N` is the number of the site (beginning with 0), and `x`, `y`, and `z` specify the location of the atom in the primitive cell in cartesian coordinates.

6. `#HAMILT`: This table defintes the Hamltonian for your system. You must provide all of the relevant information for every site defined in `#ORB`. The format is as follows

   ```
   label1   label2   x   y   z   t/mu   U
   ```

   `label1` and `label2` are integers that indicate the type of atom (if all atoms are the same, then use 0 for both). `x`, `y`, and `z` are components of the vector **r** for the interaction and are real numbers. `t/mu` and `U` are integers that indicate how many types of each quantity there are. For example, the entries that define the hopping term for the t-t' Hubbard model will look like this:

```
0   0     1.00  0.00  0.00   1  0
0   0    -1.00  0.00  0.00   1  0
0   0     0.00  1.00  0.00   1  0
0   0     0.00 -1.00  0.00   1  0
0   0     1.00  1.00  0.00   2  0
0   0    -1.00  1.00  0.00   2  0
0   0     1.00 -1.00  0.00   2  0
0   0    -1.00 -1.00  0.00   2  0
```

To specify the chemical poential `mu` and the interaction `U` at, for example, site $\mathbf{r} = 0$, one writes

```
0   0     0.00  0.00  0.00   1  1
```

Nearest-neighbor coulomb interaction can be specified for $\mathbf{r} \neq 0$. If there are multiple atom types, then one can specify additional chemical potentials or interactions.

7. `#SYMM`: There are 3 types of symmetry operations that can be specified: rotation about an axis, mirror plane, or inversion. For rotation about an axis, the convention is

   ```
   CN  x  y  z  x1  y1  z1
   ```

   where 2π/`N` is the rotation angle, `x`, `y`, and `z` specify a point belonging to the axis in cartesian coordinates, and `x1`, `y1`, and `z1` specify the axis diretion in cartesian coordinates.

   For a mirror plane operation, the format is

   ```
   D  x  y  z  x1  y1  z1
   ```

   where `x`, `y`, and `z` specify a point belonging to the plane in cartesian coordinates and `x1`, `y1`, and `z1` specify the direction normal to the plane in cartesian coordinates.

   For inversion $\mathbf{r} \to -\mathbf{r}$, the format is

   ```
   I  x  y  z
   ```

   where `x`, `y`, and `z` specify the position of the inversion point in cartesian coordinates.

8. `#PHASE`: An `NDIM`×`NDIM` integer array. Defines the primitive cell needed to describe the phase. From `NDIM` = 1, array will always be 1.

   Next, the atomic positions and their phases need to be specified. The format is

   ```
   label  x  y  z  phase
   ```

   where `label` should be consistent with the ones uses when specifying `#ORB`. For a bipartite system, `phase`= ±1.

9. `#BONDS`: In this table, the types of bonding between particles are specified. The format is

   ```
   label1  label2  x  y  z  # bond_labels
   ```

   where `label1` and `label2` indicate the atom types and `x`, `y`, and `z` specify the real space vector separating the two particles in the bond. `bond_labels`

10. #PAIR: This table specifies the different types of pairing to be measured. It uses linear combinations of the #BOND table in its definitions of the pairings. If this table is not included, QUEST will attempt to determine pairing on its own. The first line is a list of the different types of bonds (n_{bond}) by their labels.

    The remaining lines specify the types of pairing. There will be $n_{\text{bond}}+1$ entries per line. The first entry will be the label for the pairing, and the remaining entries will be real coefficients for each bond. Here we show an example the entire table with d-wave pairing (taken from square.geom).

```
#PAIR
          1     2     -2    3     -3     4     -4     5     -5
D-Wave: 0.0   1.0   1.0   0.0   0.0   -1.0   -1.0   0.0   0.0
```

11. #END: Specifies the end of the .geom file.

Appendix B has an example of general geometry definition for a 4×4 square lattice.

5 Adding 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.)

5.1 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 4.

5.2 Binning

In order to reduce correlation and bias, 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`.

5.3 Statistics

There are two special properties for the physical measurements produced by 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`.

5.4 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.[4] 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.

[4] see section 3.2 for more details.

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`.

5.5 Printing

QUEST has two subroutines that prints 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.

6 Statistics

6.1 Basics

In a Monte Carlo simulation, we often like to compute expectation value of an observable O with respect to a distribution $P(\mathbf{x})$

$$\langle O \rangle = \int d\mathbf{x}\, P(\mathbf{x})\, O(\mathbf{x}), \tag{1}$$

where $P(\mathbf{x}) \geq 0$ and satisfies

$$\int d\mathbf{x}\, P(\mathbf{x}) = 1. \tag{2}$$

If instead we only have an unnormalized distribution $p(\mathbf{x})$, one replace $p(\mathbf{x})$ with

$$P(\mathbf{x}) = \frac{p(\mathbf{x})}{Z}, \quad \text{where } Z = \int d\mathbf{x}\, p(\mathbf{x}). \tag{3}$$

To compute $\langle O \rangle$ by Monte Carlo, we draw N samples $\mathbf{x}_i$, $i = 1, 2, \ldots, N$ from $P(\mathbf{x})$. Assuming $\mathbf{x}_i$'s are independent, then the integral Eq. (1) is approximated by

$$\langle O \rangle \sim O_N = \frac{1}{N} \sum_{i=1}^{N} O(\mathbf{x}_i). \tag{4}$$

In the limit of large N, the central limit theorem tells us that the probability $\mathcal{P}(O_N)$ of obtaining a given value O_N is a Gaussian distribution with variance

$$\sigma^2 = \frac{1}{N} \left(\langle O^2 \rangle - \langle O \rangle^2 \right). \tag{5}$$

In practice we usually do not know $P(\mathbf{x})$. The variance σ is then estimated by

$$\sigma_N^2 = \frac{1}{N} \left(\frac{1}{N} \sum_{i=1}^{N} O(\mathbf{x}_i)^2 - O_N^2 \right). \tag{6}$$

While Eq. (6) provides a practical way of estimating statistical error, it is not particularly useful when one would like to estimate the variance of some function $f(\langle O\rangle)$ of $\langle O\rangle$. We could not simply define $f_i = f(O(\mathbf{x}_i))$ and use them as measurements of f. For example let x_i be a random number drawn uniformly from the interval $[0, 1]$ and $f(x) = 1/x$. Therefore we have $f(\langle x\rangle) = 0.5$, but $\langle f(x)\rangle = \infty$. The situation becomes cumbersome when f depends on more than one observables $f(O_1, O_2, \ldots)$. In these cases, one could use jackknife or bootstrap resampling methods to systematically estimate the variance of f.

6.2 Jackknife

We first consider f as a function of a single observable O_N. For a given independent sample $O(\mathbf{x}_i)$, $i = 1, 2, \ldots, N$, we define N jackknife averages as

$$O_i^J = \frac{1}{N-1} \sum_{\substack{j=1 \\ j\neq i}}^{N} O(\mathbf{x}_j). \tag{7}$$

In other words, O_i^J is the average of all measurements except $O(\mathbf{x}_i)$. Similarly, we define N jackknife samples of f

$$f_i^J = f(O_i^J). \tag{8}$$

Then the jackknife estimate of $f(O_N)$ is

$$f(O_N) \sim \overline{f^J} = \frac{1}{N} \sum_{i=1}^{N} f_i^J. \tag{9}$$

The uncertainty of this estimation is given by

$$\sigma_f = \sqrt{N}\, \sigma_{f^J}, \quad \text{where } \sigma_{f^J}^2 = \frac{1}{N} \sum_{i=1}^{N} \left(f_i^J\right)^2 - \left(\overline{f^J}\right)^2. \tag{10}$$

It is straightforward to show that if $f(O(\mathbf{x}_i)) = O(\mathbf{x}_i)$, the jackknife estimation returns to Eq. (4) and (6) in the limit of large N

$$\overline{f^J} = O_N, \tag{11}$$

$$\sigma_f^2 = \frac{N}{N-1}\, \sigma_N^2. \tag{12}$$

If f depends on several observables $f(O_{N,1}, O_{N,2}, \ldots)$, its jackknife estimate can still be calculated using Eq. (9). But now we have

$$f_i^J = f(O_{i,1}^J, O_{i,2}^J, \ldots), \tag{13}$$

where

$$O_{i,k}^J = \frac{1}{N-1} \sum_{\substack{j=1 \\ j\neq i}}^{N} O_k(\mathbf{x}_j). \tag{14}$$

6.3 Bootstrap

An alternative method for determining variance of f is the bootstrap resampling method. QUEST does not implement bootstrap to do statistics. We mention the method here only for completeness. The bootstrap method is quite similar to jackknife. The difference lies in the algorithm of resampling the data. Again, consider an independent set of measurements sample $O(\mathbf{x}_i)$, $i = 1, 2, \ldots, N$. We generate N_B sets of bootstrap "measurements" by sampling the original $O(\mathbf{x}_i)$'s *with replacement*. In other words, we do not avoid double or multiple sampling in this process. Each bootstrap sample contains exactly N data points, and its average is estimated by

$$O_i^B = \frac{1}{N} \sum_{k=1}^{N} O(\mathbf{x}_{i_k}). \tag{15}$$

Then the bootstrap estimate of f and its variance are given by

$$\overline{f^B} = \frac{1}{N_B} \sum_{i=1}^{N_B} f_i^B, \quad \text{where } f_i^B = f(O_i^B), \tag{16}$$

$$\sigma_B^2 = \frac{1}{N_B} \sum_{i=1}^{N_B} \left(f_i^B\right)^2 - \left(\overline{f^B}\right)^2 \tag{17}$$

Typically $N_B \sim \mathcal{O}(10^2)$ or $\mathcal{O}(10^3)$ is often used.

A Free form geometry definition of a 4×4 square lattice

```
name  = 2D Square Lattice (Nx=  4, Ny=  4) total sites=   16
nSite =    16
nWave =     9
dim   =     2
    4
    4

n_t   =     1
T     =    64
    2    1    1
    4    1    1
    5    1    1
   13    1    1
    1    2    1
    3    2    1
    6    2    1
   14    2    1
    2    3    1
    4    3    1
    7    3    1
   15    3    1
    1    4    1
    3    4    1
    8    4    1
   16    4    1
    1    5    1
    6    5    1
    8    5    1
    9    5    1
    2    6    1
    5    6    1
    7    6    1
   10    6    1
    3    7    1
    6    7    1
    8    7    1
   11    7    1
    4    8    1
    5    8    1
    7    8    1
   12    8    1
    5    9    1
   10    9    1
   12    9    1
   13    9    1
    6   10    1
    9   10    1
   11   10    1
   14   10    1
    7   11    1
   10   11    1
   12   11    1
   15   11    1
    8   12    1
    9   12    1
   11   12    1
   16   12    1
    1   13    1
    9   13    1
   14   13    1
   16   13    1
    2   14    1
   10   14    1
```

```
   13   14    1
   15   14    1
    3   15    1
   11   15    1
   14   15    1
   16   15    1
    4   16    1
   12   16    1
   13   16    1
   15   16    1

n_b   =     9
B     =   144
    1    1    5
    2    1    8
    4    1    2
    5    1    6
    6    1    9
    8    1    3
   13    1    4
   14    1    7
   16    1    1
    1    2    2
    2    2    5
    3    2    8
    5    2    3
    6    2    6
    7    2    9
   13    2    1
   14    2    4
   15    2    7
    2    3    2
    3    3    5
    4    3    8
    6    3    3
    7    3    6
    8    3    9
   14    3    1
   15    3    4
   16    3    7
    1    4    8
    3    4    2
    4    4    5
    5    4    9
    7    4    3
    8    4    6
   13    4    7
   15    4    1
   16    4    4
    1    5    4
    2    5    7
    4    5    1
    5    5    5
    6    5    8
    8    5    2
    9    5    6
   10    5    9
   12    5    3
    1    6    1
    2    6    4
    3    6    7
    5    6    2
    6    6    5
    7    6    8
    9    6    3
```

10	6	6
11	6	9
2	7	1
3	7	4
4	7	7
6	7	2
7	7	5
8	7	8
10	7	3
11	7	6
12	7	9
1	8	7
3	8	1
4	8	4
5	8	8
7	8	2
8	8	5
9	8	9
11	8	3
12	8	6
5	9	4
6	9	7
8	9	1
9	9	5
10	9	8
12	9	2
13	9	6
14	9	9
16	9	3
5	10	1
6	10	4
7	10	7
9	10	2
10	10	5
11	10	8
13	10	3
14	10	6
15	10	9
6	11	1
7	11	4
8	11	7
10	11	2
11	11	5
12	11	8
14	11	3
15	11	6
16	11	9
5	12	7
7	12	1
8	12	4
9	12	8
11	12	2
12	12	5
13	12	9
15	12	3
16	12	6
1	13	6
2	13	9
4	13	3
9	13	4
10	13	7
12	13	1
13	13	5
14	13	8
16	13	2

```
     1   14    3
     2   14    6
     3   14    9
     9   14    1
    10   14    4
    11   14    7
    13   14    2
    14   14    5
    15   14    8
     2   15    3
     3   15    6
     4   15    9
    10   15    1
    11   15    4
    12   15    7
    14   15    2
    15   15    5
    16   15    8
     1   16    9
     3   16    3
     4   16    6
     9   16    7
    11   16    1
    12   16    4
    13   16    8
    15   16    2
    16   16    5

nClass=     9
D     =   256
     1    1    1
     1    2    2
     1    3    3
     1    4    2
     1    5    4
     1    6    5
     1    7    6
     1    8    5
     1    9    7
     1   10    8
     1   11    9
     1   12    8
     1   13    4
     1   14    5
     1   15    6
     1   16    5
     2    1    2
     2    2    1
     2    3    2
     2    4    3
     2    5    5
     2    6    4
     2    7    5
     2    8    6
     2    9    8
     2   10    7
     2   11    8
     2   12    9
     2   13    5
     2   14    4
     2   15    5
     2   16    6
     3    1    3
     3    2    2
     3    3    1
```

```
3    4    2
3    5    6
3    6    5
3    7    4
3    8    5
3    9    9
3   10    8
3   11    7
3   12    8
3   13    6
3   14    5
3   15    4
3   16    5
4    1    2
4    2    3
4    3    2
4    4    1
4    5    5
4    6    6
4    7    5
4    8    4
4    9    8
4   10    9
4   11    8
4   12    7
4   13    5
4   14    6
4   15    5
4   16    4
5    1    4
5    2    5
5    3    6
5    4    5
5    5    1
5    6    2
5    7    3
5    8    2
5    9    4
5   10    5
5   11    6
5   12    5
5   13    7
5   14    8
5   15    9
5   16    8
6    1    5
6    2    4
6    3    5
6    4    6
6    5    2
6    6    1
6    7    2
6    8    3
6    9    5
6   10    4
6   11    5
6   12    6
6   13    8
6   14    7
6   15    8
6   16    9
7    1    6
7    2    5
7    3    4
7    4    5
```

```
 7    5    3
 7    6    2
 7    7    1
 7    8    2
 7    9    6
 7   10    5
 7   11    4
 7   12    5
 7   13    9
 7   14    8
 7   15    7
 7   16    8
 8    1    5
 8    2    6
 8    3    5
 8    4    4
 8    5    2
 8    6    3
 8    7    2
 8    8    1
 8    9    5
 8   10    6
 8   11    5
 8   12    4
 8   13    8
 8   14    9
 8   15    8
 8   16    7
 9    1    7
 9    2    8
 9    3    9
 9    4    8
 9    5    4
 9    6    5
 9    7    6
 9    8    5
 9    9    1
 9   10    2
 9   11    3
 9   12    2
 9   13    4
 9   14    5
 9   15    6
 9   16    5
10    1    8
10    2    7
10    3    8
10    4    9
10    5    5
10    6    4
10    7    5
10    8    6
10    9    2
10   10    1
10   11    2
10   12    3
10   13    5
10   14    4
10   15    5
10   16    6
11    1    9
11    2    8
11    3    7
11    4    8
11    5    6
```

```
11    6    5
11    7    4
11    8    5
11    9    3
11   10    2
11   11    1
11   12    2
11   13    6
11   14    5
11   15    4
11   16    5
12    1    8
12    2    9
12    3    8
12    4    7
12    5    5
12    6    6
12    7    5
12    8    4
12    9    2
12   10    3
12   11    2
12   12    1
12   13    5
12   14    6
12   15    5
12   16    4
13    1    4
13    2    5
13    3    6
13    4    5
13    5    7
13    6    8
13    7    9
13    8    8
13    9    4
13   10    5
13   11    6
13   12    5
13   13    1
13   14    2
13   15    3
13   16    2
14    1    5
14    2    4
14    3    5
14    4    6
14    5    8
14    6    7
14    7    8
14    8    9
14    9    5
14   10    4
14   11    5
14   12    6
14   13    2
14   14    1
14   15    2
14   16    3
15    1    6
15    2    5
15    3    4
15    4    5
15    5    9
15    6    8
```

```
   15    7    7
   15    8    8
   15    9    6
   15   10    5
   15   11    4
   15   12    5
   15   13    3
   15   14    2
   15   15    1
   15   16    2
   16    1    5
   16    2    6
   16    3    5
   16    4    4
   16    5    8
   16    6    9
   16    7    8
   16    8    7
   16    9    5
   16   10    6
   16   11    5
   16   12    4
   16   13    2
   16   14    3
   16   15    2
   16   16    1

cLabel =     9
   0  0
   1  0
   2  0
   0  1
   1  1
   2  1
   0  2
   1  2
   2  2

P      =    16
  -1.00000000000000
   1.00000000000000
  -1.00000000000000
   1.00000000000000
   1.00000000000000
  -1.00000000000000
   1.00000000000000
  -1.00000000000000
  -1.00000000000000
   1.00000000000000
  -1.00000000000000
   1.00000000000000
   1.00000000000000
  -1.00000000000000
   1.00000000000000
  -1.00000000000000

nWave =     9
W =    81
    1    1     0.00000000
    1    2     0.00000000
    1    3     0.00000000
    1    4     0.50000000
    1    5    -0.50000000
    1    6     0.00000000
    1    7     0.00000000
```

```
1    8    -0.50000000
1    9     0.00000000
2    1     0.00000000
2    2     0.50000000
2    3    -0.50000000
2    4     0.00000000
2    5     0.00000000
2    6     0.00000000
2    7    -0.50000000
2    8     0.00000000
2    9     0.00000000
3    1     0.00000000
3    2     0.00000000
3    3     0.00000000
3    4     0.50000000
3    5     0.50000000
3    6     0.00000000
3    7     0.00000000
3    8     0.00000000
3    9    -0.50000000
4    1     0.00000000
4    2     0.50000000
4    3     0.50000000
4    4     0.00000000
4    5     0.00000000
4    6    -0.50000000
4    7     0.00000000
4    8     0.00000000
4    9     0.00000000
5    1     1.00000000
5    2     0.00000000
5    3     0.00000000
5    4     0.00000000
5    5     0.00000000
5    6     0.00000000
5    7     0.00000000
5    8     0.00000000
5    9     0.00000000
6    1     0.00000000
6    2     0.50000000
6    3     0.50000000
6    4     0.00000000
6    5     0.00000000
6    6     0.50000000
6    7     0.00000000
6    8     0.00000000
6    9     0.00000000
7    1     0.00000000
7    2     0.00000000
7    3     0.00000000
7    4     0.50000000
7    5     0.50000000
7    6     0.00000000
7    7     0.00000000
7    8     0.00000000
7    9     0.50000000
8    1     0.00000000
8    2     0.50000000
8    3    -0.50000000
8    4     0.00000000
8    5     0.00000000
8    6     0.00000000
8    7     0.50000000
8    8     0.00000000
8    9     0.00000000
```

```
    9    1     0.00000000
    9    2     0.00000000
    9    3     0.00000000
    9    4     0.50000000
    9    5    -0.50000000
    9    6     0.00000000
    9    7     0.00000000
    9    8     0.50000000
    9    9     0.00000000

wLabel  =     9
           S-Wave :
          SX-Wave :
           D-Wave :
         SXX-Wave :
         DXX-Wave :
          PX-Wave :
          PY-Wave :
         PXY-Wave :
         PYX-Wave :

FT =    81
           1           1   1.00000000000000
           1           2   2.00000000000000
           1           3   1.00000000000000
           1           4   2.00000000000000
           1           5   4.00000000000000
           1           6   2.00000000000000
           1           7   1.00000000000000
           1           8   2.00000000000000
           1           9   1.00000000000000
           2           1   1.00000000000000
           2           2  8.742278000372451E-008
           2           3 -0.999999999999996
           2           4   2.00000000000000
           2           5  1.748455600074490E-007
           2           6  -1.99999999999999
           2           7   1.00000000000000
           2           8  8.742278000372451E-008
           2           9 -0.999999999999996
           3           1   1.00000000000000
           3           2  -1.99999999999996
           3           3  0.999999999999985
           3           4   2.00000000000000
           3           5  -3.99999999999992
           3           6   1.99999999999997
           3           7   1.00000000000000
           3           8  -1.99999999999996
           3           9  0.999999999999985
           4           1   1.00000000000000
           4           2   2.00000000000000
           4           3   1.00000000000000
           4           4  8.742278000372451E-008
           4           5  1.748455600074490E-007
           4           6  8.742278000372451E-008
           4           7 -0.999999999999996
           4           8  -1.99999999999999
           4           9 -0.999999999999996
           5           1   1.00000000000000
           5           2  8.742278000372451E-008
           5           3 -0.999999999999996
           5           4  8.742278000372451E-008
           5           5  7.549516567451064E-015
           5           6 -8.742278000372347E-008
           5           7 -0.999999999999996
```

```
5           8 -8.742278000372347E-008
5           9  0.999999999999985
6           1   1.00000000000000
6           2  -1.99999999999996
6           3  0.999999999999985
6           4  8.742278000372451E-008
6           5 -1.748455600074430E-007
6           6  8.742278000372181E-008
6           7 -0.999999999999996
6           8   1.99999999999992
6           9 -0.999999999999966
7           1   1.00000000000000
7           2   2.00000000000000
7           3   1.00000000000000
7           4  -1.99999999999996
7           5  -3.99999999999992
7           6  -1.99999999999996
7           7  0.999999999999985
7           8   1.99999999999997
7           9  0.999999999999985
8           1   1.00000000000000
8           2  8.742278000372451E-008
8           3 -0.999999999999996
8           4  -1.99999999999996
8           5 -1.748455600074430E-007
8           6   1.99999999999992
8           7  0.999999999999985
8           8  8.742278000372181E-008
8           9 -0.999999999999966
9           1   1.00000000000000
9           2  -1.99999999999996
9           3  0.999999999999985
9           4  -1.99999999999996
9           5   3.99999999999972
9           6  -1.99999999999987
9           7  0.999999999999985
9           8  -1.99999999999987
9           9  0.999999999999939
```

B General geometry definition of a 4×4 square lattice

```
#NDIM
2
#PRIM
1.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
#SUPER
4 0
0 4
#K-POINT
0.0 0.0
#ORB
s0  0.0d0  0.0d0  0.0d0 #0
#HAMILT
0 0 1.0 0.0 0.0 1.0 0.0 0.0   # hopping in x direction
0 0 0.0 1.0 0.0 1.0 0.0 0.0   # hopping in y direction
0 0 0.0 0.0 0.0 0.0 0.0 4.0   # onsite U=4 and mu=0
#SYMM
C4 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 1.d0
D  0.0d0 0.0d0 0.0d0 1.0d0 0.0d0 0.d0
D  0.0d0 0.0d0 0.0d0 0.0d0 1.0d0 0.d0
#PHASE
1 1
-1 1
s0 0.0 0.0 0.0 1.0
s0 0.0 1.0 0.0 -1.0
#XBONDS
0 0  0.0  0.0  0.0 # 1
0 0  1.0  0.0  0.0
0 0  2.0  0.0  0.0
0 0  1.0  1.0  0.0
0 0  2.0  1.0  0.0
#XBONDS
0 0  1.0  0.0  0.0 # 1 -1
0 0  2.0  0.0  0.0 # 2 -2
0 0  2.0  1.0  0.0 # 3 -3
#XBONDS
0 0  0.0  0.0  0.0 # 1
0 0  1.0  0.0  0.0 # 2 -2
0 0  1.0  1.0  0.0 # 3 -3
#XBONDS
0 0  0.0  0.0  0.0 # 1
0 0  1.0  0.0  0.0 # 2 -2
0 0  1.0  1.0  0.0 # 3 -3
0 0  0.0  1.0  0.0 # 4 -4
0 0 -1.0  1.0  0.0 # 5 -5
#XPAIR
             1      2    -2     3     -3      4     -4     5    -5
D-Wave:      0.0    1.0   1.0   0.0    0.0   -1.0   -1.0   0.0   0.0
#END
```