Reference Manual for Interacting Quantum Impurity Systems Simulating Toolkit

Draft Version January 10, 2015

Written by:

Li Huang, **Yilin Wang**, **Zi Yang Meng**, and **Liang Du**

*i*QIST Developer Team

Core Developers:

Li Huang[†] and **Yilin Wang**[‡]

Key Contributors:

Zi Yang Meng[‡,♭] and **Liang Du**[♯]

Directors and Supervisors:

Philipp Werner[†] and **Xi Dai**[‡]

[†]Department of Physics, University of Fribourg, 1700 Fribourg, Switzerland

[‡]Beijing National Laboratory for Condensed Matter Physics, and
Institute of Physics, Chinese Academy of Sciences, Beijing 100190, China

[♭]Department of Physics, University of Toronto, Toronto, Ontario M5S 1A7, Canada

[♯]Department of Physics, The University of Texas at Austin, Austin, Texas 78712, USA

To my lovely wife X. Zhao

L. H

To my lovely girlfriend X.Y. Mao

Y.L. Wang

Contents

List of Figures

List of Tables

Acknowledges

What's new?

How to use this reference manual?

Chapter 1

INTRODUCTION

1.1 What's *i*QIST ?

The Interacting Quantum Impurity Solver Toolkit (dubbed *i*QIST) is an open source software package aiming to provide a full, reliable, flexible, and powerful tool chain for various quantum impurity models. It contains a few continuous-time quantum Monte Carlo impurity solvers (hybridization expansion version), a Hirsch-Fye quantum Monte Carlo impurity solver, and numerous prep-processed and post-processed tools. The *i*QIST is an all-in-one package. With it you can solve quantum impurity models and anaylyze the calculated results easily and efficiently.

1.2 Motivation

Dynamical mean-field theory (DMFT)[?] and its cluster extensions[? ?] play a very important role in contemporary studies of correlated electron systems. The broad applications of this technique range from the study of Mott transitions[? ?], unconventional superconductivity in Cu- and Fe-based superconductors[? ? ? ? ? ?], and non-Fermi liquid behaviors[? ? ?], to the investigation of anomalous transport properties of transition metal oxides[?]. For many of these applications, DMFT is the currently most powerful and reliable (sometimes the only) technique available and has in many cases produced new physical insights. Furthermore, the combination of *ab initio* calculation method (such as density function theory) with DMFT[?] allows to compute the subtle electronic properties of realistic correlated materials, includ-

ing partially filled $3d$- and $4d$-electron transition metal oxides, where lattice, spin and orbital degrees of freedom all coupled[?] .

The key idea of DMFT is to map the original correlated lattice model into a quantum impurity model whose mean-field bath is determined self-consistently[? ? ?] . Thus, the central task of a DMFT simulation becomes the numerical solution of the quantum impurity problem. During the past several decades, many methods have been tested as impurity solvers, including the exact diagonalization (ED)[?] , equation of motion (EOM)[?] , Hubbard-I approximation (HIA)[?] , iterative perturbation theory (IPT)[?] , non-crossing approximation (NCA)[? ? ?] , fluctuation-exchange approximation (FLEX)[? ?] , and quantum Monte Carlo (QMC)[? ?] . Among the methods listed above, the QMC method has several very important advantages, which makes it so far the most flexible and widely used impurity solver. First, it is based on the imaginary time action, in which the infinite bath has been integrated out. Second, it can treat arbitrary couplings, and can thus be applied to all kinds of phases including the metallic phase, insulating state, and phases with spontaneous symmetry breaking. Third, the QMC method is numerically exact with a "controlled" numerical error. In other words, by increasing the computational effort the numerical error of the QMC simulation can be systematically reduced. For these reasons, the QMC algorithm is considered as the method of choice for many applications.

Several QMC impurity solvers have been developed in the past three decades. An important innovation was the Hirsch-Fye QMC (HF-QMC) impurity solver[? ?] , in which the time axis is divided into small time steps and the interaction term in the Hamiltonian is decoupled on each time step by means of a discrete Hubbard-Stratonovich auxiliary field. HF-QMC has been widely used in the early studies of DMFT[? ? ?] , but is limited by the discretization on the time axis and also by the form of the electronic interactions (usually only density-density interactions can be treated). Recently, a new class of more powerful and versatile QMC impurity solvers, continuous-time quantum Monte Carlo (CT-QMC) algorithms, have been invented[? ? ? ? ? ?] . In the CT-QMC impurity solvers, the partition function of the quantum impurity problem is diagrammatically expanded, and then the diagrammatic expansion series is evaluated by stochastic Monte Carlo sampling. The continuous-time nature of the algorithm means that operators can be placed at any arbitrary position on the imaginary time interval, so that time discretization errors can be completely avoided. Depending on how the diagrammatic expansion is performed, the CT-QMC approach can be further divided into interaction expansion (or weak coupling) CT-QMC (CT-INT)[?] , auxiliary field CT-QMC (CT-AUX)[?] , and hybridization expansion (or strong coupling) CT-QMC (CT-HYB)[? ? ?] .

At present, the CT-HYB is the most popular and powerful impurity solver, since it can be used to solve multi-orbital impurity model with general interactions at low temperature[?] . In single-site DMFT calculations, the computational efficiency of CT-HYB is much higher than that of CT-INT and HF-QMC, especially when the interactions are strong. However, in order to solve more complicated quantum impurity models (for example, five-band or seven-band impurity model with general interactions and spin-orbital coupling) efficiently, further improvements of the CT-HYB impurity solvers are needed. In recent years many tricks and algorithms have been developed to increase the efficiency and accuracy of original CT-HYB impurity solver, such as the truncation approximation[?] , Krylov subspace iteration[?] , orthogonal polynomial representation[?,?] , PS quantum number[?] , lazy trace evaluation and skip listing methods[?] , and matrix product state implementation[?] . As the state-of-the-art CT-HYB impurity solvers become more sophisticated and specialized, it is not easy anymore to master all their facets and build ones implementations from scratch. Hence, we believe that it is a good time to provide a CT-HYB software package for the DMFT community such that researchers can focus more on the physical questions, instead of spending much time on (re-)implementing efficient codes. In fact, there are some valuable efforts in this direction, such as TRIQS[?] , ALPS[?] , w2dynamics[?] , Haule's code[?] , etc. However, a flexible, extensible, and highly efficient CT-HYB impurity solver is still lacking. The purpose of this reference manual is to present our solution -- the open source software package *i*QIST -- which contains several well-implemented and thoroughly tested modern CT-HYB impurity solvers, and the corresponding pre- and post-processing tools.

1.3 Software architecture

To solve a quantum impurity model is not a straightforward job. Besides the necessary quantum impurity solvers, we need several auxiliary programs or tools. The *i*QIST is an all-in-one software package, which can be used to solve a broad range of quantum impurity problems. Thus, it is not surprising that *i*QIST is a collection of various codes and scripts. The core components contain about 50000 lines of code.

The software architecture of *i*QIST is slightly involved. In Fig. 1.1, we use a layer model to illustrate it. The bottom layer is the operating system (OS). In principle, the *i*QIST is OS-independent. It can run properly on top of Unix/Linux, Mac OS X, FreeBSD, and Windows. The second layer is the system layer, which contains highly optimized linear algebra math libraries (such as BLAS and LAPACK)

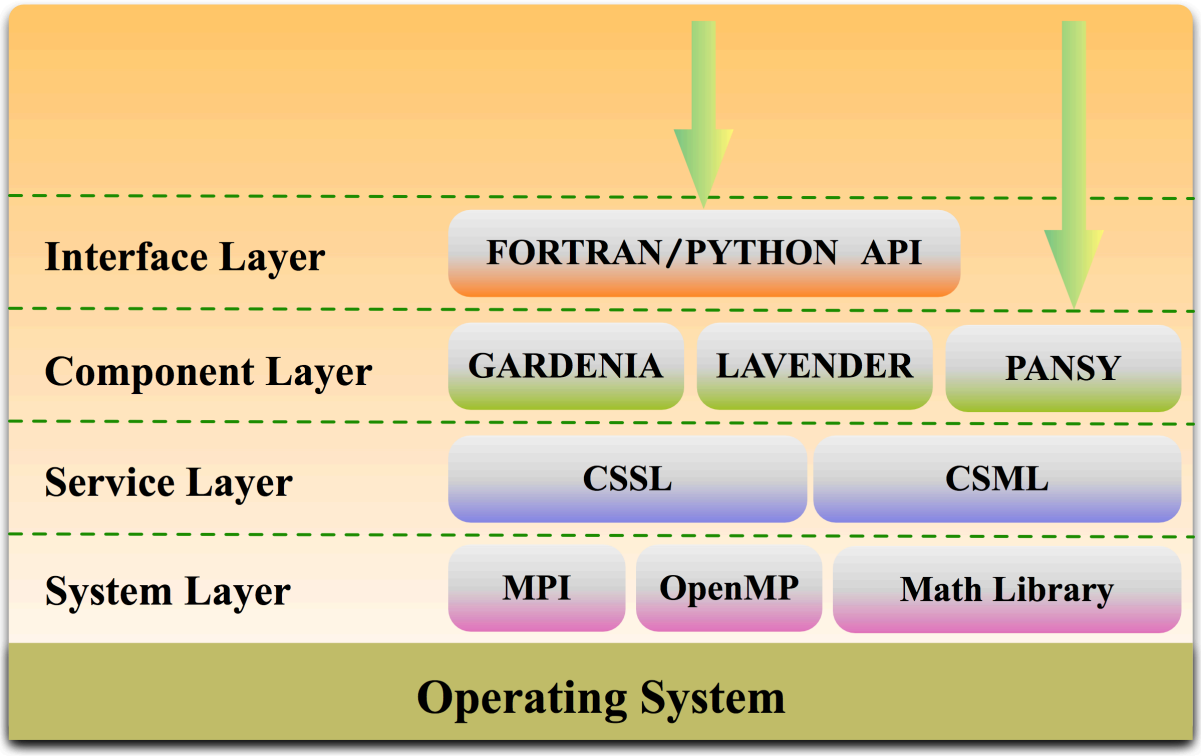

Figure 1.1: The hierarchical structure of the *i*QIST software package. Note that in the component layer, not all of the components are listed due to space limitations. See the main text for detailed explanations.

and parallelism supports (such as MPI and OpenMP). The third layer is the service layer. In this layer, we implemented some commonly used modules and subroutines. They are named as common service subroutine library (CSSL) and common service module library (CSML), respectively. They provide a useful interface between the system layer and the component layer and facilitate the development of core components. The features of CSSL and CSML include basic data structures (stack and linked list), random number generators, spare matrix manipulations, linear algebra operations, string processing, linear interpolation, numerical integration, and fast Fourier transformation (FFT), etc.

The core part of *i*QIST is in the fourth layer -- the component layer -- which contains various impurity solvers as shown in Fig. 1.2. At present, *i*QIST contains ten different components, including *A*zalea , *G*ardenia , *N*arcissus , *B*egonia , *L*avender , *P*ansy , *M*anjushaka , *D*aisy , *J*asmine , and *H*ibiscus . Here, *A*zalea , *G*ardenia , *N*arcissus , *B*egonia , *L*avender , *P*ansy , and *M*anjushaka are all CT-HYB components (as shown in the LHS of Fig. 1.2), and *D*aisy is a HF-QMC impurity solver component. *J*asmine is an atomic eigenvalue solver. *H*ibiscus is a collection of several

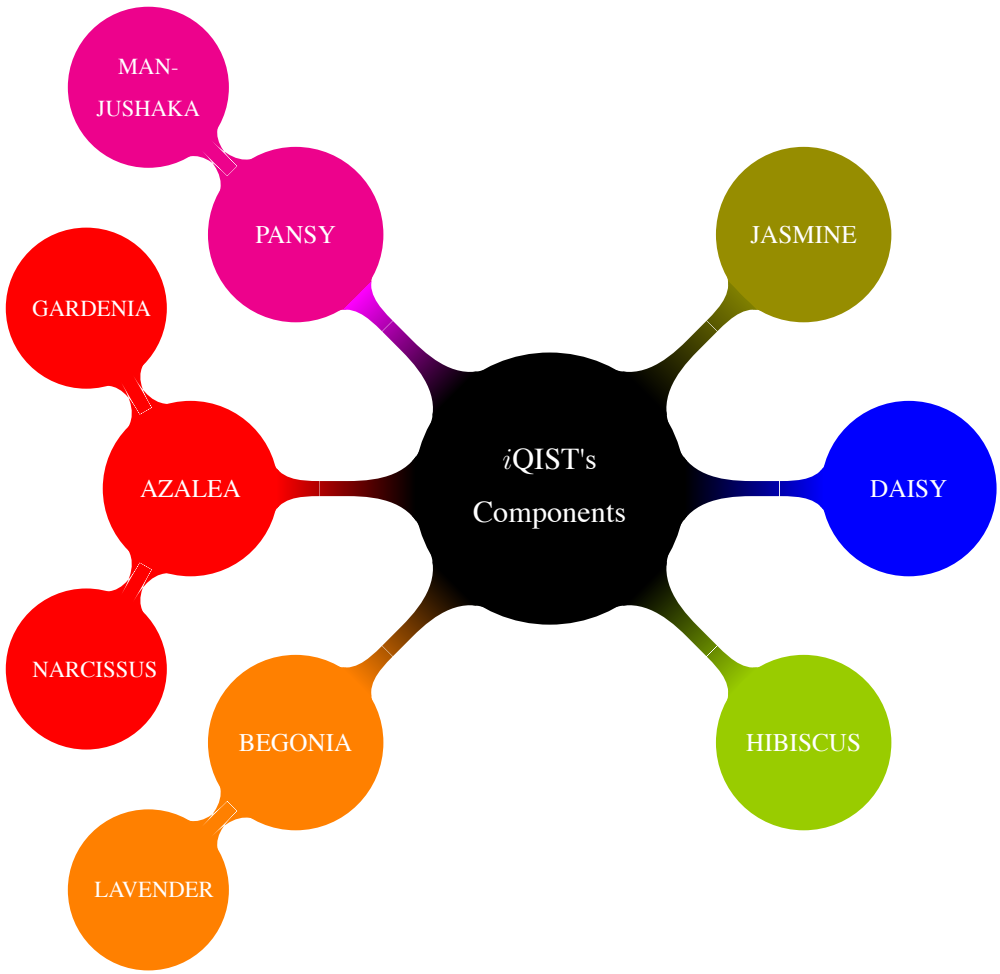

Figure 1.2: Schematic picture for the *i*QIST 's components. Components on the LHS are CT-HYB solvers, $\mathcal{J}$asmine is the atomic eigenvalue solver, *D*aisy is a HF-QMC solver, and $\mathcal{H}$ibiscus contains other pre-processing and post-processing tools.

pre- and post-processing tools, including maximum entropy method, stochastic analytical continuation, Padé approximation, and Kramers-Kronig transformation, etc. For more details about these components, please consult the following sections.

The top layer is the interface layer or user layer. On the one hand, users can execute the *i*QIST 's components directly as usual. On the other hand, they can also invoke *i*QIST 's components from other languages. The role of *i*QIST 's components becomes a library or subroutine. To achieve this goal, in the interface layer, we offer the Fortran/C/C++/Python language bindings for most of the *i*QIST 's components, so that the users can develop their own codes on top of *i*QIST and consider it as a

computational engine in black box.

1.4 Main features

Table 1.1: The models supported by CT-HYB impurity solvers in the *i*QIST software package. In this table, the CT-HYB impurity solvers are presented using the first capital letter of their names. For example, A denotes the *A*zalea component.

Models	CT-HYB
Density-density interaction	A, G, N, B, L, P, M
General Coulomb interaction (Slater or Kanamori schemes)	B, L, P, M
Spin-orbital coupling (SOC) interaction	B, L, P, M
Crystal field splitting	A, G, N, B, L, P, M
Hubbard-Holstein model	N
Frequency-dependent interaction	N

Table 1.2: The measurement tricks used by CT-HYB impurity solvers in the *i*QIST software package.

Measurement tricks	CT-HYB
Orthogonal polynomial representation (Legendre and Chebyshev polynomials)	G, N, L, M
Kernel polynomial representation (Legendre and Chebyshev polynomials)	G, N, L, M
Improved estimator for self-energy	G, N

As mentioned before, *i*QIST software packages contain several CT-HYB impurity solvers (as schematically shown in Fig. 1.2). In this subsection, in order to help the users to choose a suitable CT-HYB impurity solver, we briefly discuss their main features, pros, and cons. The main results are also summarized in Tab. 1.1-1.4 for a quick query.

Table 1.3: The trace algorithms supported by CT-HYB impurity solvers in the *i*QIST software package.

Trace algorithms	CT-HYB
Segment representation algorithm for density-density interaction	`A, G, N`
Divide-and-conquer algorithm	`B, L, P, M`
Sparse matrix multiplication	`B, L`
Good quantum numbers	`P, M`
Skip listing trick	`M`
Lazy trace evaluation	`M`
Dynamical truncation approximation	`M`

When the Coulomb interaction term in the local Hamiltonian $H_{\rm loc}$ only retains density-density terms, $H_{\rm loc}$ becomes a diagonal matrix in the occupation number basis. In this case, the CT-HYB impurity solver is extremely efficient if the so-called segment picture (or segment algorithm)[??] is adopted. Thus, we implemented the segment algorithm in the *Azalea*, *Gardenia*, and *Narcissus* components.

In the *Azalea* component, we only implemented the basic segment algorithm and very limited physical observables are measured. It is the simplest and the most efficient algorithm. In fact, it is the development prototype of the other CT-HYB components, and usually used to test some experimental features. In the *Gardenia* component, we add more features on the basis of the *Azalea* component. For example, we can use the orthogonal polynomial technique to improve the numerical accuracy and suppress stochastic noise in the Green's function[?]. The self-energy function can be measured with the improved estimator algorithm[??]. More single-particle and two-particle correlation functions are measured. Though *Gardenia* is much more powerful than *Azalea*, it is a bit less efficient. The features of the *Narcissus* component are almost the same as those of the *Gardenia* component. In addition, it can be used to deal with dynamically screened interactions[??]. In other words, the Coulomb interaction U needs not to be a static value any more, but can be frequency-dependent. Thus, it is used for example in extended-DMFT

Table 1.4: The observables measured by CT-HYB impurity solvers in the *i*QIST software package.

Physical observables	CT-HYB
Single-particle Green's function $G(\tau)$	A, G, N, B, L, P, M
Single-particle Green's function $G(i\omega_n)$	A, G, N, B, L, P, M
Two-particle correlation function $\chi(\omega, \omega', \nu)$	G, N, L, M
Local irreducible vertex function $\Gamma(\omega, \omega', \nu)$	G, N, L, M
Pair susceptibility $\Gamma(\omega, \omega', \nu)$	G, N, L, M
Self-energy function $\Sigma(i\omega_n)$	A, G, N, B, L, P, M
Histogram of perturbation expansion order	A, G, N, B, L, P, M
Kinetic and potential energies	A, G, N, B, L, P, M
(Double) occupation numbers, magnetic moment	A, G, N, B, L, P, M
Atomic state probability	A, G, N, B, L, P, M
Spin-spin correlation function	G, N
Orbital-orbital correlation function	G, N
Autocorrelation function and autocorrelation time	G, N, L, M

calculations[?] . Note that since the Hubbard-Holstein model can be mapped in DMFT onto a dynamical Anderson impurity model[?] , it can be solved using the $\mathcal{N}$arcissus component as well.

When the local Hamiltonian H_{loc} contains general Coulomb interaction terms, the general matrix formulation[? ?] , which is implemented in the $\mathcal{B}$egonia , $\mathcal{L}$avender , $\mathcal{P}$ansy , and $\mathcal{M}$anjushaka components, should be used. Each of these components has its own features and targets specific systems.

In the $\mathcal{B}$egonia component, we implemented the direct matrix-matrix multiplications algorithm. We adopted the divide-and-conquer scheme and sparse matrix tricks to speed up the calculation. This component can be used to deal with the impurity models with up to 3 bands with fairly good efficiency. However, it is not suitable for 5- and 7-band systems. In the $\mathcal{L}$avender component, we implemented all the same algorithms as int the $\mathcal{B}$egonia component. Besides, we implemented the orthogonal polynomial representation to improve the measurement quality of physical quantities. Some two-particle quantities are also measured. This component, as well, can only be used to conduct calculations for $1 \sim 3$ bands systems. But, it can produce measurements of very high quality with small additional cost.

In the $\mathcal{P}$ansy component, we considered the symmetries of H_{loc} and applied the GQN trick to accelerate the evaluation of local trace. This algorithm is general and doesn't depend on any details of the GQNs, so it can support all the GQNs schemes which fulfill the conditions discussed in Sec. 10.2.5. We also adopted the divide-and-conquer algorithm to speed it up further. This component can be used to study various impurity models ranging from 1-band to 5-band with fairly good efficiency. However, it is still not suitable for 7-band models. In the $\mathcal{M}$anjushaka component, we implemented all the same algorithms as the $\mathcal{P}$ansy component. Besides, we implemented the lazy trace evaluation[?] to speed up the Monte Carlo sampling process. It can gain quite high efficiency, and is especially useful in the low temperature region. We also implemented a smart algorithm to truncate some high energy states dynamically in the Hilbert space of H_{loc} to speed up the trace evaluation further. This algorithm is very important and efficient (in some sense it is necessary) for dealing with 7-band systems. We implemented the orthogonal polynomial representation to improve the measurements of key observables as well. By using all of these tricks, the computational efficiency of the $\mathcal{M}$anjushaka component for multi-orbital impurity models with general Coulomb interaction is unprecedentedly high. We believe that it can be used to study nearly all of the quantum impurity systems ranging from 1-band to 7-band.

1.5 Roadmap

Although proven to be very versatile in applications and efficient in performance, the *i*QIST project is still a work in progress and the development will continue. The future developments of the *i*QIST project are likely to be along the following directions.

As the study of interacting electronic systems is moving towards treating their correlated multi-band nature in more realistic fashion (5- or 7-bands, SOC included, competing multi-orbital interactions, etc.), it is important to develop even more efficient and optimized CT-HYB impurity solvers. An effective way to reduce the average size of the matrices used during the calculation is to fully consider the point group symmetry of the impurity model, which provides more GQNs to the problem. The corresponding coding work has already been started by some of the authors.

Recent developments in condensed matter theories need to be added into the features of the *i*QIST software package. For example, the measurement of entanglement entropy in realistic correlated fermion systems[? ?] will be considered, with which one will be able to explore and discovery more symmetry protected topological states and even interaction-driven topological orders that might exist in nature[? ?].

The two-particle correlation functions (susceptibilities) contain more information than the single-particle quantities, but the DMFT formalism is only self-consistent at the single-particle level. To conduct a calculation which is self-consistent both at the single- and two-particle levels is the next step in the CT-HYB/DMFT simulations. The DMFT + Parquet scheme present in Sec. 10.1.5 is the first step to incorporate correlation effects at the two-particle level beyond single-site DMFT, but it is only self-consistent at the two-particle level, and in many occasions we only perform one-shot simulation at the two-particle level due to numerical difficulties. To be fully self-consistent among single- and two-particle quantities, one still needs to employ the Schwinger-Dyson equation to feed the two-particle information back to the single-particle quantities[? ?]. This will also be a further development of the *i*QIST software package.

Instead of using single- and two-particle diagrammatic relations to capture the spatial correlation effects, one can also develop cluster CT-QMC impurity solvers, such that the spatial correlation within the cluster can be captured exactly. While in one-band models and a few two-band models the cluster CT-QMC impurity solvers are available[? ? ? ? ?], generic cluster CT-QMC impurity solvers which take care of both the multi-orbital interactions within each cluster site and the spatial correlations between the cluster sites are still missing. This is also an arena for future developments.

In the end, we would like to emphasize that *i*QIST is an open initiative and the feedback and contributions from the community are very welcome.

1.6 Policy

License

The *i*QIST software package is released under the General Public Licence 3.0 (GPL) or later version.

Registration

Registration is not obligated of course. But if you can send your email address to us, we can inform you on time once the new *i*QIST is released.

Technical support

We are sorry. We DO NOT provide any technical supports now. If you meet some problems when you are using *i*QIST . You can write a letter to us. But we can not guarantee that we will reply you always.

Feedback

If you have any suggestions, comments, successful stories, or critisms, welcome! Please write a letter to us.

Contribution

If you find a bug, want to patch it, or want to contribute your codes to *i*QIST , great! Please write a letter to us as soon as possible.

Donation

We DO NOT need any donations now and in the future.

Contact

Dr. Li HUANG (email: huangli712 at gmail.com)

Dr. Yilin WANG (email: qhwyl2006 at 126.com)

Citation

It is really appreciated if you can cite the following paper when you would like to publish your great works by using *i*QIST . It is really important for us. But, of course, it is also not obligated.

iQIST: An open source continuous-time quantum Monte Carlo impurity solver toolkit

Li Huang, Yilin Wang, Zi Yang Meng, Liang Du, Philipp Werner and Xi Dai

arXiv:1409.7573 (2014)

Chapter 2

INSTALLATION

2.1 Obtain

The *i*QIST is an open source free software package. We release it under the General Public Licence 3.0 (GPL). The readers who are interested in it can write a letter to the authors to request an electronic copy of the newest version of *i*QIST :

email: huangli712@gmail.com,

or they can download it directly from the public code repository:

http://bitbucket.org/huangli712/iqist.

2.2 Uncompress

The downloaded *i*QIST software package is likely a compressed file with zip or tar.gz suffix. The users should uncompress it at first. For examples:

```
$ tar xvfz iqist.tar.gz.
```

2.3 Direcrory structures

The directory structure of the *i*QIST software package is shown in Fig. 2.1.

COPYING: GNU General Public License Version 3.

LICENSE: Copyright declaration.

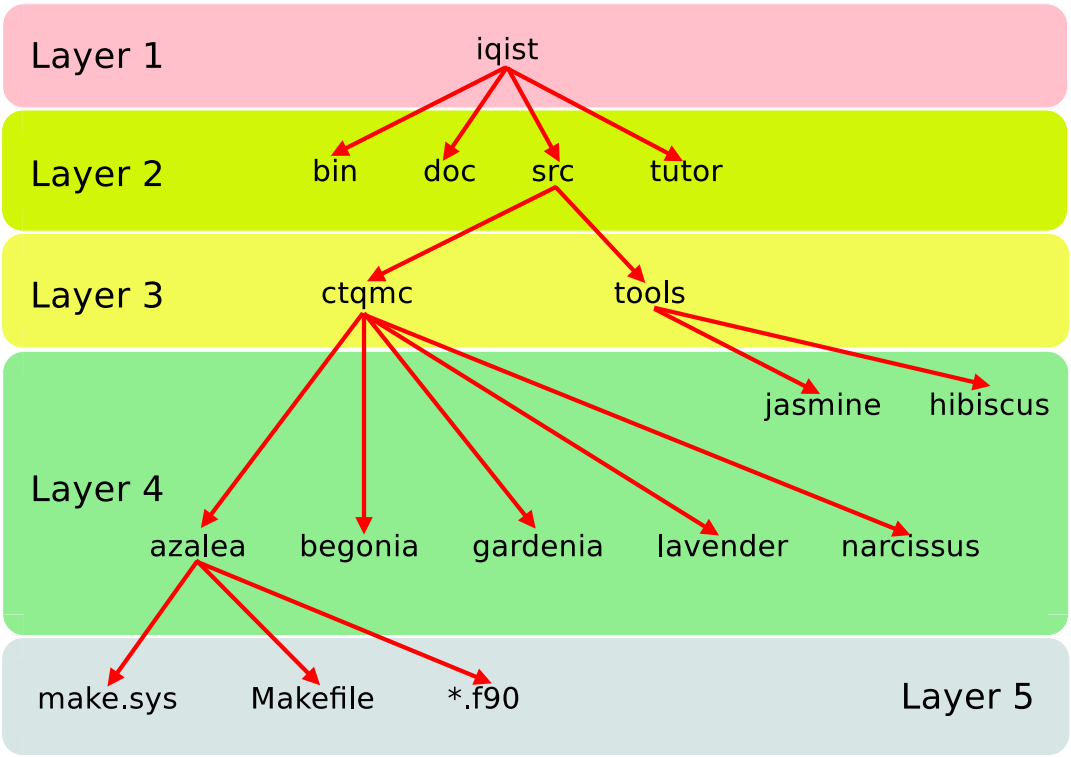

Figure 2.1: The directory structure of *i*QIST software package.

README.md: Readme.

RELEASES.md: Release notes.

iqist/bin: Used to save the executable programs.

iqist/doc: Documentation. The reference manual is in the manual directory. The document in the chinese directory is a bit outdated and will be removed in the future.

iqist/src: All of the source codes are here.

iqist/tutor: Tutorial materials are here.

iqist/working: Collection of typical input files which are used to benchmark *i*QIST .

Now let's make a dive into the iqist/src directory.

iqist/src/build: Top Makefile is here.

iqist/src/common: Common Service Subroutine Library (CSSL) and Common Service Module Library (CSML).

iqist/src/ctqmc/api: Application Programming Interface (API) for CT-QMC impurity solver.

iqist/src/ctqmc/azalea: The *A*zalea component.

iqist/src/ctqmc/begonia: The *B*egonia component.

iqist/src/ctqmc/gardenia: The *G*ardenia component.

iqist/src/ctqmc/lavender: The *L*avender component.

iqist/src/ctqmc/manjushaka: The *M*anjushaka component.

iqist/src/ctqmc/narcissus: The *N*arcissus component.

iqist/src/ctqmc/pansy: The *P*ansy component.

iqist/src/hfqmc/daisy: The *D*aisy component.

iqist/src/tools/hibiscus: The *H*ibiscus component.

iqist/src/tools/jasmine: The *J*asmine component.

2.4 Compiling environment

In order to compile and install *i*QIST correctly, you should ensure the following softwares are correctly installed and configured in your OS.

Fortran Compiler

Most of the *i*QIST 's components are written in Fortran 90 language, so in order to compile *i*QIST , you need a Fortran 90 compiler. The components in *i*QIST can be successfully compiled using a recent

Intel Fortran compiler. In fact, during the development of *i*QIST , we always used the Intel Fortran compiler. We can not guarantee that it can be compiled successfully by the other Fortran compilers, such as gfortran, PGI Fortran compiler, etc.

MPI

All of the CT-QMC and HF-QMC impurity solvers in *i*QIST are fully parallelized with MPI. In order to obtain parallelized versions of quantum impurity solvers, some kinds of MPI implementations must be installed beforehand. Most of the MPI implementations, such as MPICH, MVAPICH, OpenMPI and Intel MPI are compatible with *i*QIST .

BLAS

The *i*QIST depends on the BLAS library. As for the BLAS implementation, we strongly recommend OpenBLAS. The GotoBLAS2 is somewhat outdated and is not maintained any more, so we do not recommend it.

LAPACK

The *i*QIST depends on the LAPACK library. For the LAPACK, the Intel Math Kernel Library is a good candidate. Of course, it is also possible to use the linear algebra library provided by the operating system, for example, the vecLib Framework in the Mac OS X.

Python

Some post-processing scripts contained in the $\mathcal{H}$ibiscus component are developed using the Python language. In order to execute these scripts or use the Python language binding for *i*QIST , the users should install Python 2.x. Furthermore, the newest numpy, scipy, and f2py packages are also necessary.

2.5 Compiling system

We use the make utility to build the *i*QIST software package. If you go through the directory structure of *i*QIST , you can find many Makefiles in different directories (for examples, iqist/src/ctqmc/api and iqist/src/hfqmc/daisy directories). For a given directory, if there is a Makefile in it, then you can always input the "make" command to build the code, the "make clean" command to clean binary objects, the "make clean-dat" command to clean the data files in it.

The top Makefile is in the iqist/src/build directory. In this directory, once you input the "make" command, a menu will be displayed in the terminal. Then you can follow the instructions to select suitable building jobs. In this directory, there is also a make.sys file which is used to configure the whole

compiling system. Before you type any "make" commands, you have to edit the make.sys file at first to setup the compiling environment correctly. At least, you must setup the Fortran compiler, MPI compiler, BLAS and LAPACK libraries manually. All of the Makefile files depend on this make.sys. So be careful.

Next, we will explain the options in the make.sys file in details:

F90 = mpif90

Intel Fortran compiler. It can be 'mpif90' or 'ifort'. If it is mpif90, the internal compiler invoked by mpif90 must be ifort.

LINKER = $(F90)

Linker. Here it is the same with compiler. Do not change it.

ARCHIVER = ar -ruv

Archiver. It is used to pack the objects into a library. Do not modify it for ever.

MPI = -DMPI

Specify whether MPI is enable. If you want to compile a sequential code, please comment it out with '#' symbol and then setup F90 to 'ifort'. We strongly suggest to compile MPI parallelized codes.

OMP = #-openmp

Specify whether OpenMP is enable. If you want to disable it, please comment it out. In default it is disabled. So far OpenMP was not used by any components.

API = #-DAPI -DF2PY

Specify whether we want to obtain a library version of *i*QIST . If it is enabled, what you can obtain is a library and you can write python or fortran codes to use it, instead of a standard executable program. At default it is disabled. If only the API macro is defined, the Fortran API will be generated. If both the API and F2PY macros are defined, the Python API and (slightly different) Fortran API will be generated. If only the F2PY macro is defined, it is invalid.

FPP = -fpp

Specify whether the fortran preprocessor (FPP) is used. It must be enabled or else the *i*QIST can not be compiled correctly.

CPP = $(FPP) $(MPI) $(OMP) $(API)

Collection of preprocessor directives. Do not modify it unless you are an expert of *i*QIST .

GPROF = #-pg

Specify whether the code profiling should be done. If it is enabled, then after the code is finished, a gmon.out file will be generated. You can use the gprof tool to analyze the runtime information and find

out the hotspot of the code. It is not wise to enable it to build a production code, because it will decrease the efficiency greatly.

CHECK = -warn all #-check all -traceback -g

Used to specify what types of check should be done. '-warn all' means the check is done in compiling. '-check all' means the check will be done in running. '-traceback' enable us to track the exact position (line number and file name) where the error occurs. '-g' enable the compiler to generate debug information and embed them into the final program. Note that '-check all', '-traceback', and '-g' opinions will decrease the efficiency greatly.

CDUMP = -vec-report2 -openmp-report2 -nogen-interfaces

Specify whether the compiler will output useful optimization information in compiling.

LEVEL = -O3 -xHost -unroll-aggressive -align all

Collection of optimization opinions. '-O3' means the highest optimization. '-xHost' enables the compiler to generate the most suitable code for the current computer architecture. '-unroll-aggressive' means using aggressive method to unroll the loop structures. '-align all' means to align the arrays, structures, etc. Please modify them only if you are an expert of Intel Fortran Compiler and you know what you are doing.

MARCH = -march=corei7-avx # core2 corei7 corei7-avx core-avx-i core-avx2

Used to specify the instruction sets that the current system can support. 'core2' is the safest choice and it works always. But it may be not the best. Please modify it only when you understand what you are doing.

FFLAGS = -c $(CPP) $(CHECK) $(CDUMP) $(LEVEL) $(MARCH) $(GPROF)

Collection of Fortran compiler opinions. Do not modify them for ever.

LFLAGS = -static -Wl,-no_pie $(OMP) $(GPROF)

Collection of linker opinions. '-Wl,-no_pie' is useful when you are using Mac Os X system. Do not modify them unless you know what you are doing.

LIBS = -L/opt/intel/mkl/lib -lmkl_core -lmkl_sequential -lmkl_rt

Specify the external libraries. Now the *i*QIST software package depends on LAPACK and BLAS heavily. To achieve good performance, the highly optimized LAPACK and BLAS implementations are essential. Here we want to recommend the OpenBLAS and Intel MKL.

MPIL = -L/usr/local/openmpi/lib -lmpi_mpifh

Specify the external mpi libraries for fortran binding. It is useful when you are compiling the python

application programming interface using f2py (pyiqist and pydaisy). The f2py will automatically use the 'bare' intel fortran compiler (ifort) to link and generate the python modules (*.so). If you do not specify MPIL, then the python modules will not work properly.

2.6 Build full iQIST at one step

After a few minutes (depending on the performance of compiling platform), the iQIST is ready for use. Note that all of the executable programs will be copied into the iqist/bin directory automatically. Please add this directory into the system environment variable PATH.

2.7 Build quantum impurity solvers

2.8 Build auxiliary tools

2.9 Build documents

2.10 Build library

2.11 Build application programming interfaces

Chapter 3

RUNNING

3.1 Configure your system

3.2 Create input files

3.3 Execute codes

To use iQIST is easy. At first, since there are several CT-HYB impurity solvers in the package and their features and efficiencies are somewhat different, it is the user's responsibility to choose suitable CT-HYB components to deal with the impurity problem at hand. Second, the iQIST is in essence a computational engine, so users have to write scripts or programs to execute the selected CT-HYB impurity solver directly or to call it using the application programming interface. For example, if the users want to conduct CT-HYB/DMFT calculations, they must implement the DMFT self-consistent equation by themselves. Third, an important task is to prepare proper input data for the selected CT-HYB impurity solver. The optional input for the CT-HYB impurity solver is the hybridization function [$\Delta(\tau)$ or $\Delta(i\omega_n)$], impurity level ($E_{\alpha\beta}$), interaction parameters (U, J, and μ), etc. If users do not provide them to the impurity solver, it will use the default settings automatically. Specifically, if the Coulomb interaction matrix is general, users should use the `JASMINE` component to solve the local atomic problem at first to generate the necessary eigenvalues and eigenvectors. Fourth, execute the CT-HYB impurity solver. Finally, when the calculations are finished, users can use the tools contained in the `HIBISCUS`

component to post-process the output data, such as the imaginary-time Green's function $G(\tau)$, Matsubara self-energy function $\Sigma(i\omega_n)$, and other physical observables. For more details, please refer to the user guide of iQIST.

3.4 Monitor and profile

Chapter 4

STANDARD INPUT FILES

4.1 solver.ctqmc.in

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

4.2 solver.eimp.in

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

4.3 solver.hyb.in

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

4.4 solver.anydos.in

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

4.5 solver.ktau.in

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

4.6 atom.cix

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

Chapter 5

STANDARD OUTPUT FILES

5.1 Terminal output

5.1.1 out.dat

5.2 File output

5.2.1 solver.green.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.2 solver.green.bin

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.3 solver.weiss.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.4 solver.hybrid.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.5 solver.grn.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.6 solver.wss.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.7 solver.hyb.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.8 solver.sgm.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.9 solver.hub.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.10 solver.nmat.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.11 solver.schi.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.12 solver.ochi.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.13 solver.twop.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.14 solver.vrtx.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.15 solver.pair.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.16 solver.hist.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.17 solver.prob.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.18 solver.kernel.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

5.2.19 solver.status.dat

CONTENT:

COMPONENT:

DATA FORMAT:

DATA VISUALIZATION:

COMMENT:

Chapter 6

PARAMETERS

6.1 solver.ctqmc.in and solver.hfqmc.in

In this section, we will introduce all of the parameters which can be used in the solver.ctqmc.in (for Azalea , Gardenia , Narcissus , Begonia , Lavender , Pansy , and Manjushaka components) and solver.hfqmc.in (for Daisy component) files. For more information, the uses can also refer to the comments in corresponding ctqmc_control.f90 file.

6.1.1 Jp

DEFINITION: Strength of pair-hopping term in the interaction term.

DATATYPE: real(dp).

DEFAULT: 0.0_dp.

COMPONENT: ALL, except for Daisy .

BEHAVIOR: Actually, this parameter is not used by the code internally. It will be outputted by the impurity solver as a reference.

COMMENT: NONE.

6.1.2 Js

DEFINITION: Strength of spin-flip term in the interaction term.

DATATYPE: real(dp).

DEFAULT: 0.0_dp.

COMPONENT: ALL, except for $\mathcal{D}$aisy .

BEHAVIOR: Actually, this parameter is not used by the code internally. It will be outputted by the impurity solver as a reference.

COMMENT: NONE.

6.1.3 Jz

DEFINITION: Hund's exchange coupling constant.

DATATYPE: real(dp).

DEFAULT: 0.0_dp.

COMPONENT: ALL.

BEHAVIOR: In the $\mathcal{A}$zalea , $\mathcal{G}$ardenia , $\mathcal{N}$arcissus , and $\mathcal{D}$aisy codes, it is used to build the Coulomb interaction matrix. However, in the $\mathcal{B}$egonia , $\mathcal{L}$avender , $\mathcal{P}$ansy , and $\mathcal{M}$anjushaka codes, this parameter is not used internally. It will be outputted by the impurity solver as a reference.

COMMENT: As for single-band model, $J_z = J_s = J_p = 0.0$. The condition $U_c = U_v - 2J_z$ should be always fulfilled.

6.1.4 U

DEFINITION: Averaged Coulomb interaction.

DATATYPE: real(dp).

DEFAULT: 4.0_dp.

COMPONENT: ALL, except for $\mathcal{D}$aisy .

BEHAVIOR: Actually, this parameter is not used by the code internally. It will be outputted by the impurity solver as a reference.

COMMENT: Usually, we let $U = U_c$.

6.1.5 Uc

DEFINITION: Inter-orbital Coulomb interaction.

DATATYPE: real(dp).

DEFAULT: 4.0_dp.

COMPONENT: ALL.

BEHAVIOR: In the *Azalea* , *Gardenia* , *Narcissus* , and *Daisy* codes, it is used to build the Coulomb interaction matrix. However, in the *Begonia* , *Lavender* , *Pansy* , and *Manjushaka* codes, this parameter is not used internally. It will be outputted by the impurity solver as a reference.

COMMENT: The condition $U_c = U_v - 2J_z$ should be always fulfilled.

6.1.6 Uv

DEFINITION: Intra-orbital Coulomb interaction.

DATATYPE: real(dp).

DEFAULT: 4.0_dp.

COMPONENT: ALL, except for *Daisy* .

BEHAVIOR: Actually, this parameter is not used by the code internally. It will be outputted by the impurity solver as a reference.

COMMENT: The condition $U_c = U_v - 2J_z$ should be always fulfilled.

6.1.7 alpha

DEFINITION: Mixing parameter α, used to mix the old and new self-energy function to produce a newer one.

DATATYPE: real(dp).

DEFAULT: 0.7_dp.

COMPONENT: ALL.

BEHAVIOR: $\Sigma_{\text{new}} = \alpha\Sigma_{\text{new}} + (1.0 - \alpha)\Sigma_{\text{old}}$

COMMENT: If you have trouble in obtaining convergence, you can decrease it to 0.5_dp or even 0.2_dp. If you are doing one-shot calculation (isscf = 1), this parameter is useless.

6.1.8 beta

DEFINITION: Inversion of temperature $\beta(= 1.0/\text{T})$. Its unit is eV^{-1}.

DATATYPE: real(dp).

DEFAULT: 8.0_dp

COMPONENT: ALL.

BEHAVIOR: The impurity solver will use it to determine the system temperature.

COMMENT: The real temperature is 11604.505008098/β Kelvin.

6.1.9 chgrd

DEFINITION: Number of linear grid points in [-1,1].

DATATYPE: integer.

DEFAULT: 20001.

COMPONENT: Only for $\mathcal{G}$ardenia , $\mathcal{N}$arcissus , $\mathcal{L}$avender , and $\mathcal{M}$anjushaka .

BEHAVIOR: The second kind Chebyshev orthogonal polynomials are defined in a linear grid in [-1,1]. And the number of grid points are controlled by this parameter.

COMMENT: Only when isort = 3 or isort = 6 this parameter is useful. The 20001 is an optimal value for chgrd. It is not suggested to modified it. About the isort parameter, please refer to Sec. 6.1.13.

As for the applications of orthogonal polynomials in CT-QMC impurity solver, please refer to Phys. Rev. B 84. 075145 (2011) and Phys. Rev. B 85, 205106 (2012).

6.1.10 chmax

DEFINITION: The maximum allowable expansion order for the second kind Chebyshev orthogonal polynomials. It must be greater than 2.

DATATYPE: integer.

DEFAULT: 32.

COMPONENT: Only for $\mathcal{G}$ardenia , $\mathcal{N}$arcissus , $\mathcal{L}$avender , and $\mathcal{M}$anjushaka .

BEHAVIOR: The parameter is used as a cutoff to limit the maximum expansion order for the second kind Chebyshev orthogonal polynomials.

COMMENT: Only when isort = 3 or isort = 6 this parameter is useful. How to choose a suitable chmax parameters is a tricky job. If chmax is too small, the calculated results won't be accurate. If chmax is too large, the so-called Gibbs oscillation will occur dramatically. According to our experiences, 32 or 48 may be a reasonable choice. There is no upper limit. But the larger chmax, the heavier computational burden. About the isort parameter, please refer to Sec. 6.1.13.

As for the applications of orthogonal polynomials in CT-QMC impurity solver, please refer to Phys. Rev. B 84. 075145 (2011) and Phys. Rev. B 85, 205106 (2012).

6.1.11 ifast

DEFINITION: Key control flag. It is used to select the suitable algorithm to calculate the operator trace.

DATATYPE: integer.

DEFAULT: 1.

COMPONENT: Only for $\mathcal{M}$anjushaka .

BEHAVIOR: There are three possible values for ifast parameter so far:

- ifast = 1, the divide-and-conquer algorithm will be used. At this time, you have to adjust npart parameter carefully to obtain good computational efficiency.
- ifast = 2, the classic time evolution algorithm will be used. WARNING: this algorithm is not implemented in the public version. We are sorry for that.
- ifast = 3, the skip listing algorithm will be used. WARNING: this algorithm is not implemented in the public version. We are sorry for that.

All in all, now only ifast = 1 is valid. Even you set ifast = 2 or ifast = 3, $\mathcal{M}$anjushaka code will still use the divide-and-conquer algorithm to calculate the operator trace.

COMMENT: In the $\mathcal{M}$anjushaka code, the Lazy trace evaluation trick is always used.

As for the divide-and-conquer algorithm and the classic time evolution algorithm, please refer to Emanuel Gull's PhD thesis. As for the skip listing algorithm, please refer to Phys. Rev. B. 90 075149 (2014).

6.1.12 isbin

DEFINITION: Key control flag. It is used to determine whether we need to use the data binning mode.

DATATYPE: integer.

DEFAULT: 2.

COMPONENT: ALL.

BEHAVIOR: There are three possible values for isbin parameter so far:

- isbin = 1, normal mode.

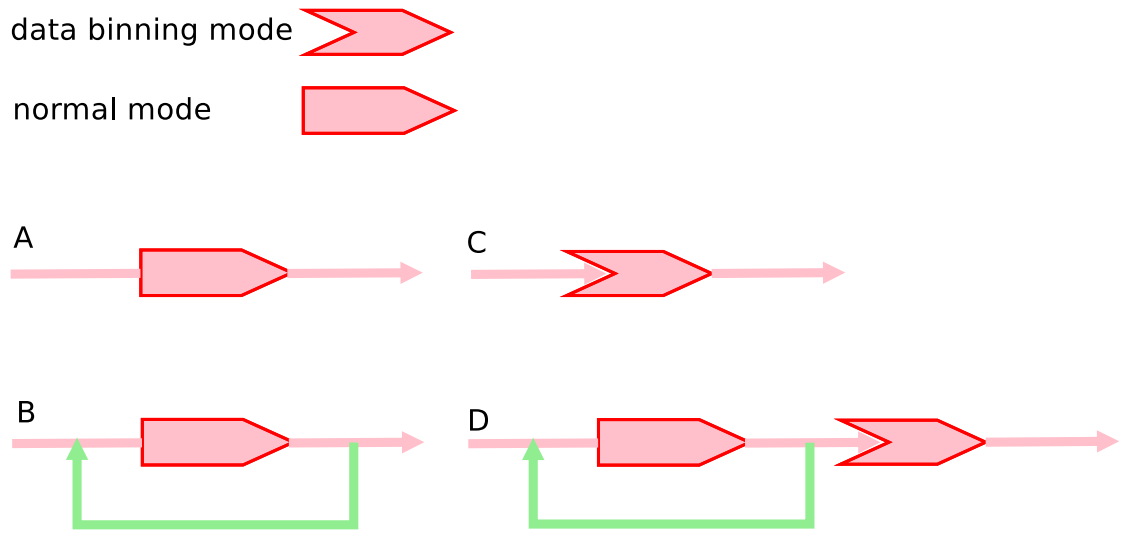

Figure 6.1: The four running modes for quantum impurity solvers.

- isbin = 2, data binning mode.

When the impurity solver is in the data binning mode, the current iteration number (iter, it is a internal variable) will be fixed to 999, nsweep and nwrite parameters will be increased by a factor of ten. The imaginary-time Green's function $G(\tau)$ will be outputted periodically to solver.green.bin.X files (where X denotes the index of data bins, from 1 to nsweep/nwrite), instead of solver.green.dat file. The post-processing tools will be used to deal with these files to do analytical continuation calculation.

COMMENT: From the combinations of isscf and isbin parameters, we can reach the following four running modes:

- A: isscf = 1, isbin = 1. The impurity solver is called only once. No DMFT self-consistent calculation. No data binning mode.
- B: isscf = 2, isbin = 1. The impurity solves is called periodically in the DMFT self-consistent calculation. Once the convergence is reached, the calculation will be stopped. No data binning mode.
- C: isscf = 1, isbin = 2. The impurity solver is called only once in the data binning mode. No DMFT self-consistent calculation.
- D: isscf = 2, isbin = 2. The impurity solves is called periodically in the DMFT self-consistent calculation. During the iterations, the impurity solver is in normal mode. After the convergence is

reached, the impurity solver will be called once again. But at this time, the data binning mode is activated automatically.

Please see Fig. 6.1 for a intuitive view of the four running modes. If we want to perform LDA + DMFT calculations, mode A will be a good choice. Mode B is often used to solve model Hamiltonians iteratively and quickly to judge whether the results are reasonable. Once the results are reasonable and what we expect, and we want more accurate data, then we can use mode C. Mode D can be viewed as a combination of mode B and mode C, it is seldom used.

6.1.13 isort

DEFINITION: Key control flag. It is used to determine whether we should use the orthogonal polynomials trick to improve the accuracy and suppress the numerical noises.

DATATYPE: integer.

DEFAULT: 1.

COMPONENT: Only for $\mathcal{G}$ardenia , $\mathcal{N}$arcissus , $\mathcal{L}$avender , and $\mathcal{M}$anjushaka .

BEHAVIOR: There are six possible values for isort parameter so far:

- isort = 1, the standard method is used to measure $G(\tau)$.
- isort = 2, the Legendre orthogonal polynomials trick is used to measure $G(\tau)$.
- isort = 3, the Chebyshev orthogonal polynomials trick is used to measure $G(\tau)$.
- isort = 4, the standard method is used to measure $G(\tau)$ and $F(\tau)$
- isort = 5, the Legendre orthogonal polynomials trick is used to measure $G(\tau)$ and $F(\tau)$.
- isort = 6, the Chebyshev orthogonal polynomials trick is used to measure $G(\tau)$ and $F(\tau)$.

Here $G(\tau)$ is the imaginary-time Green's function and $F(\tau)$ auxiliary imaginary-time function which can be used to calculate $\Sigma(i\omega_n)$ analytically[1].

COMMENT:

[1] Only when the Coulomb interaction matrix is density-density type, and $\mathcal{G}$ardenia or $\mathcal{N}$arcissus component is used.

6.1.14 isscf

DEFINITION:

DATATYPE: integer.

DEFAULT: 2.

COMPONENT: ALL.

BEHAVIOR:

- isscf = 1,
- isscf = 2,

COMMENT:

6.1.15 isscr

DEFINITION:

DATATYPE: integer.

DEFAULT: 1.

COMPONENT: Only for $\mathcal{N}$arcissus .

BEHAVIOR:

- isscr = 1
- isscr = 2
- isscr = 3
- isscr = 4
- isscr = 99

COMMENT:

6.1.16 isspn

DEFINITION:

DATATYPE: integer.

DEFAULT: 1.

COMPONENT: ALL.

BEHAVIOR:

- isspn = 1,
- isspn = 2,

COMMENT:

6.1.17 issun

DEFINITION:

DATATYPE: integer.

DEFAULT: 2.

COMPONENT: ALL.

BEHAVIOR:

- issun = 1,
- issun = 2,

COMMENT:

6.1.18 isvrt

DEFINITION:

DATATYPE: integer.

DEFAULT:

COMPONENT: Only for $\mathcal{G}$ardenia , $\mathcal{N}$arcissus , $\mathcal{L}$avender , and $\mathcal{M}$anjushaka .

BEHAVIOR:

COMMENT:

6.1.19 itrun

DEFINITION: Key control flag. It is used to control which scheme should be used to truncate the Hilbert space.

DATATYPE: integer.

DEFAULT: 1.

COMPONENT: Only for $\mathcal{M}$anjushaka .

BEHAVIOR: There are two possible values for itrun parameter so far:

- itrun = 1, no truncation.
- itrun = 2, those atomic states with low probability ($P_\Gamma < 1.0E-6$) will be truncated for next DMFT iteration.

COMMENT: This feature is experimental. Please check your calculated results carefully if you used the truncation approximation. To perform the truncation, the $\mathcal{M}$anjushaka will read the solver.prob.dat file at first to get the probability data. So if this file is not available, the truncation will not be done.

To perform truncation over occupation number, you should use the $\mathcal{J}$asmine component to generate suitable atom.cix file.

6.1.20 lc

DEFINITION: It is a model parameter for Hubbard-Holstein model or dynamical screening effect. The exact definition of lc depends on the value of isscr parameter.

DATATYPE: real(dp).

DEFAULT: 1.0_dp.

COMPONENT: Only for $\mathcal{N}$arcissus .

BEHAVIOR:

- if isscr = 1, no meaning.
- if isscr = 2, the λ parameter in the Hubbard-Holstein model.
- if isscr = 3, the λ parameter in the plasmon-pole model for dynamical screening effect.
- if isscr = 4, the α parameter in the ohmic model for dynamical screening effect.
- if isscr = 99, shift for the chemical potential and Coulomb interaction (when we consider the dynamical screening effect in the LDA + DMFT calculations for realistic materials). You can obtain this value from the output of $\mathcal{H}$ibiscus /toolbox/makescr.x.

COMMENT: Only when isscr $>$ 1, it matters. As for the isscr parameter, please refer to Sec. 6.1.15. About CT-QMC algorithm for Hubbard-Holstein model, please refer to Phys. Rev. Lett 99, 146404 (2007). About dynamical screening effect, plasmon-pole model, and ohmic mode, please refer to Phys. Rev. Lett. 104, 146401 (2010).

6.1.21 legrd

DEFINITION: Number of linear grid points in [-1,1].

DATATYPE: integer.

DEFAULT: 20001.

COMPONENT: Only for $\mathcal{G}$ardenia , $\mathcal{N}$arcissus , $\mathcal{L}$avender , and $\mathcal{M}$anjushaka .

BEHAVIOR: The Legendre orthogonal polynomials are defined in a linear grid in [-1,1]. And the number of grid points are controlled by this parameter.

COMMENT: Only when isort = 2 or isort = 5 this parameter is useful. The 20001 is an optimal value for legrd. It is not suggested to modified it. About the isort parameter, please refer to Sec. 6.1.13.

As for the applications of orthogonal polynomials in CT-QMC impurity solver, please refer to Phys. Rev. B 84. 075145 (2011) and Phys. Rev. B 85, 205106 (2012).

6.1.22 lemax

DEFINITION: The maximum allowable expansion order for the Legendre orthogonal polynomials. It must be greater than 2.

DATATYPE: integer.

DEFAULT: 32.

COMPONENT: Only for $\mathcal{G}$ardenia , $\mathcal{N}$arcissus , $\mathcal{L}$avender , and $\mathcal{M}$anjushaka .

BEHAVIOR: The parameter is used as a cutoff to limit the maximum expansion order for the Legendre orthogonal polynomials.

COMMENT: Only when isort = 2 or isort = 5 this parameter is useful. How to choose a suitable lemax parameters is a tricky job. If lemax is too small, the calculated results won't be accurate. If lemax is too large, the so-called Gibbs oscillation will occur dramatically. According to our experiences, 32 or 48 may be a reasonable choice. It is worthy to emphasis that due to the limitation of implementation, lemax must be less than 50. About the isort parameter, please refer to Sec. 6.1.13.

As for the applications of orthogonal polynomials in CT-QMC impurity solver, please refer to Phys. Rev. B 84. 075145 (2011) and Phys. Rev. B 85, 205106 (2012).

6.1.23 mfreq

DEFINITION: Number of matsubara frequency points.

DATATYPE: integer.

DEFAULT: 8193 ($= 2^{13} + 1$).

COMPONENT: ALL.

BEHAVIOR: The $G(i\omega_n)$, $\mathcal{G}(i\omega_n)$, $\Sigma(i\omega_n)$, and $\Delta(i\omega_n)$ are all defined in this matsubara frequency grid.

COMMENT: It is an optimal value. Please do not modify it.

6.1.24 mkink

DEFINITION: Maximum allowable diagrammatic perturbation order in the continuous-time quantum Monte Carlo algorithm.

DATATYPE: integer.

DEFAULT: 1024.

COMPONENT: ALL, except for $\mathcal{D}$aisy .

BEHAVIOR:

COMMENT: It is an optimal value. Please do not modify it.

6.1.25 mstep

DEFINITION:

DATATYPE: integer.

DEFAULT:

COMPONENT: Only for $\mathcal{D}$aisy .

BEHAVIOR:

COMMENT:

6.1.26 mune

DEFINITION: Chemical potential or Fermi level μ.

DATATYPE: real(dp)

DEFAULT: 2.0_dp.

COMPONENT: ALL.

BEHAVIOR:

COMMENT: For single-band Hubbard mode, the half-filling condition is $\mu = U/2.0$.

6.1.27 nband

DEFINITION: Number of bands (N_{band}).

DATATYPE: integer.

DEFAULT: 1

COMPONENT: ALL.

BEHAVIOR:

COMMENT: In *i*QIST , when we say nband, we always do not consider the freedom of spin. So for $d-$electron system, it should be a five-band model (nband = 5), while for $f-$electron, it should be a seven-band model (nband = 7).

6.1.28 nbfrq

DEFINITION: Number of bosonic frequencies.

DATATYPE: integer.

DEFAULT: 4.

COMPONENT: Only for *G*ardenia , *N*arcissus , *L*avender , and *M*anjushaka .

BEHAVIOR: The two-particle Green's function $\chi(i\omega, i\omega', i\nu)$ and vertex function $\mathcal{F}(i\omega, i\omega', i\nu)$ have three frequency indices where ω and ω' are fermionic frequencies, and ν bosonic frequency:

$$\nu_n = \frac{2n\pi}{\beta} \tag{6.1}$$

$$\omega_n = \frac{(2n+1)\pi}{\beta} \tag{6.2}$$

The nbfrq parameter is used to define and generate ν_n bosonic mesh. The corresponding ω_n fermionic mesh is defined by nffrq parameter.

COMMENT: Usually nbfrq is small (but nbfrq > 0), since the computational burden for two-particle quantities is extremely heavy.

6.1.29 ncarlo

DEFINITION:

DATATYPE: integer.

DEFAULT:

COMPONENT: ALL.

BEHAVIOR:

COMMENT:

6.1.30 ncfgs

DEFINITION: Number of atomic configurations N_{cfgs}.

DATATYPE: integer.

DEFAULT: 4.

COMPONENT: ALL.

BEHAVIOR: ncfgs = 2^{norbs}.

COMMENT: Attention, the ncfgs should be consistent with nband, norbs, and nspin. The *i*QIST will not check the correctness of them.

6.1.31 nclean

DEFINITION:

DATATYPE: integer.

DEFAULT:

COMPONENT: ALL.

BEHAVIOR:

COMMENT:

6.1.32 nffrq

DEFINITION: Number of fermionic frequencies.

DATATYPE: integer.

DEFAULT: 32.

COMPONENT: ALL.

BEHAVIOR: The two-particle Green's function $\chi(i\omega, i\omega', i\nu)$ and vertex function $\mathcal{F}(i\omega, i\omega', i\nu)$ have three frequency indices where ω and ω' are fermionic frequencies, and ν bosonic frequency:

$$\nu_n = \frac{2n\pi}{\beta} \tag{6.3}$$

$$\omega_n = \frac{(2n+1)\pi}{\beta} \tag{6.4}$$

The nffrq parameter is used to define and generate ω_n fermionic mesh. The corresponding ν_n bosonic mesh is defined by nbfrq parameter.

COMMENT: Usually nffrq is small (but nffrq > 0), since the computational burden for two-particle quantities is extremely heavy.

6.1.33 nflip

DEFINITION:

DATATYPE: integer.

DEFAULT: 2000.

COMPONENT: ALL.

BEHAVIOR:

COMMENT:

6.1.34 nfreq

DEFINITION:

DATATYPE: integer.

DEFAULT: 128.

COMPONENT: ALL.

BEHAVIOR:

COMMENT:

6.1.35 niter

DEFINITION: Number of DMFT self-consistent iterations.

DATATYPE: integer.

DEFAULT: 20.

COMPONENT: ALL.

BEHAVIOR:

COMMENT:

6.1.36 nmonte

DEFINITION:

DATATYPE: integer.

DEFAULT:

COMPONENT: ALL.

BEHAVIOR:

COMMENT:

6.1.37 norbs

DEFINITION: Number of orbitals (N_{orbs}).

DATATYPE: integer.

DEFAULT: 2.

COMPONENT: ALL.

BEHAVIOR:

COMMENT:

6.1.38 npart

DEFINITION:

DATATYPE: integer.

DEFAULT: 4.

COMPONENT: ALL.

BEHAVIOR:

COMMENT:

6.1.39 nsing

DEFINITION: Number of Ising auxiliary fields.

DATATYPE: integer.

DEFAULT:

COMPONENT: Only for *Daisy* .

BEHAVIOR:

COMMENT:

6.1.40 nspin

DEFINITION: Number of spin projections.

DATATYPE: integer.

DEFAULT: 2.

COMPONENT: ALL.

BEHAVIOR:

COMMENT: DO NOT MODIFY IT. IT MUST BE 2 ALWAYS.

6.1.41 nsweep

DEFINITION:

DATATYPE: integer.

DEFAULT:

COMPONENT: ALL.

BEHAVIOR:

COMMENT:

6.1.42 ntherm

DEFINITION: Number of thermalization (warmup) steps.

DATATYPE: integer.

DEFAULT:

COMPONENT: ALL.

BEHAVIOR:

COMMENT:

6.1.43 ntime

DEFINITION: Number of time slices.

DATATYPE: integer.

DEFAULT:

COMPONENT: ALL.

BEHAVIOR:

COMMENT:

6.1.44 nwrite

DEFINITION:

DATATYPE: integer.

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.1.45 nzero

DEFINITION:

DATATYPE: integer.

DEFAULT:

COMPONENT: Only for $\mathcal{B}$egonia and $\mathcal{L}$avender .

BEHAVIOR:

COMMENT:

6.1.46 part

DEFINITION: Hopping paramter t in the Hubbard model

DATATYPE: real(dp).

DEFAULT: 0.5_dp

COMPONENT: ALL.

BEHAVIOR: This parameter is only used to initialize the initial hybridization function $\Delta(i\omega)$ and calculate new hybridization function through the self-consistent condition ($\Delta = t^2 G$, for bethe lattice).

COMMENT: This parameter has influences on the results only when isscf = 2. About the isscf parameter, please refer to Sec. 6.1.14.

6.1.47 wc

DEFINITION:

DATATYPE: real(dp)

DEFAULT: 1.0_dp

COMPONENT: Only for $\mathcal{N}$arcissus .

BEHAVIOR:

COMMENT:

6.2 entropy.in and sac.in

6.2.1 ainit

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.2 beta

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.3 devia

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.4 eta1

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.5 eta2

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.6 legrd

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.7 lemax

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.8 ltype

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.9 nalph

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.10 nband

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.11 ndump

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.12 ngamm

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.13 ngrid

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.14 niter

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.15 norbs

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.16 nstep

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.17 ntime

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.18 ntune

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.19 ntype

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.20 nwarm

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.21 nwmax

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.22 ratio

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.23 sigma

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.2.24 wstep

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3 atom.config.in

6.3.1 Jh

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.2 Jp

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.3 Js

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.4 Jz

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.5 Uc

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.6 Ud

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.7 Uv

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.8 ibasis

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.9 icf

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.10 ictqmc

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.11 icu

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.12 isoc

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.13 lambda

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.14 mune

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.15 nband

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.16 ncfgs

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.17 nmaxi

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.18 nmini

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.19 norbs

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

6.3.20 nspin

DEFINITION:

DATATYPE:

DEFAULT:

COMPONENT:

BEHAVIOR:

COMMENT:

Chapter 7

AUXILIARY TOOLS

7.1 $\mathcal{D}$aisy component

7.2 $\mathcal{J}$asmine component

When the Coulomb interaction is general in the local Hamiltonian H_{loc}, as discussed above, we have to diagonalize H_{loc} in advance to obtain its eigenvalues, eigenvectors, and the F-matrix. In general, the local Hamiltonian is defined as

$$H_{\text{loc}} = H_{\text{int}} + H_{\text{cf}} + H_{\text{soc}}, \tag{7.1}$$

where H_{int} means the Coulomb interaction term, H_{cf} the CF splitting term, and H_{soc} the SOC interaction. The `JASMINE` component is used to solve this Hamiltonian and generate necessary inputs for some CT-HYB impurity solvers (i.e., `BEGONIA`, `LAVENDER`, `PANSY`, and `MANJUSHAKA` components).

The `JASMINE` component will build H_{loc} in the Fock representation at first. For the Coulomb interaction term H_{int}, both Kanamori parameterized and Slater parameterized forms are supported. In other words, users can use U and J, or Slater integrals F^k to define the Coulomb interaction matrix as they wish. For the CF splitting term H_{cf}, either diagonal or non-diagonal elements are accepted. The SOC term H_{soc} is defined as follows,

$$H_{\text{soc}} = \lambda \sum_i \vec{\mathbf{l}}_i \cdot \vec{\mathbf{s}}_i, \tag{7.2}$$

where λ is the strength for SOC. Note that the SOC term can only be activated for the 3-, 5-, and 7-band systems.

Next, the JASMINE component will diagonalize $H_{\rm loc}$ to get all eigenvalues and eigenvectors. There are two running modes for JASMINE. (1) It diagonalizes $H_{\rm loc}$ in the full Hilbert space directly to obtain the eigenvalues E_α and eigenvectors Γ_α, then the F-matrix is built from the eigenvectors,

$$(F_i)_{\alpha,\beta} = \langle \Gamma_\alpha | F_i | \Gamma_\beta \rangle, \tag{7.3}$$

where i is the flavor index. The eigenvalues and F-matrix will be fed into the BEGONIA and LAVENDER components as necessary input data. (2) It diagonalizes each subspace of $H_{\rm loc}$ according to the selected GQNs. Currently, four GQNs schemes for various types of $H_{\rm loc}$ are supported, which are listed in Table 10.1. JASMINE also builds indices to record the evolution sequence depicted in Eq. (10.35). According to the indices, it builds the F-matrix between two different subspaces. The eigenvalues, the indices, and the F-matrix will be collected and written into a disk file (atom.cix), which will be read by the PANSY and MANJUSHAKA components.

Apart from this, the JASMINE component will also generate the matrix elements of some physical operators, such as $\vec{L}^2$, L_z, $\vec{S}^2$, S_z, $\vec{J}^2$, and J_z, etc. They can be used by the other post-processing codes to analyze the averaged expectation value of these operators.

7.3 $\mathcal{H}$ibiscus component

In the HIBISCUS component, many auxiliary tools are provided to deal with the output data of the CT-HYB impurity solvers. Here we briefly describe some of these tools:

Maximum entropy method

In the Monte Carlo community, the maximum entropy method[?] is often used to extract the spectral function $A(\omega)$ from the imaginary time Green's function $G(\tau)$. Thus, in the HIBISCUS component, we implemented the standard maximum entropy algorithm. In the EDMFT calculations, sometimes we have to perform analytical continuation for the retarded interaction function $\mathcal{U}(i\nu)$ to obtain $\mathcal{U}(\nu)$. So we developed a modified version of the maximum entropy method to enable this calculation.

Stochastic analytical continuation

An alternative way to extract the $A(\omega)$ from $G(\tau)$ is the stochastic analytical continuation[?] . Unlike the maximum entropy method, the stochastic analytical continuation does not depend on any *a priori* parameters. It has been argued that the stochastic analytical continuation can produce more accurate spectral functions with more subtle structures. In the HIBISCUS component, we also implemented

the stochastic analytical continuation which can be viewed as a useful complementary procedure to the maximum entropy method. Since the stochastic analytical continuation is computationally much heavier than the maximum entropy method, we parallelized it with MPI and OpenMP.

Kramers-Kronig transformation

Once the analytical continuation is finished, we can obtain the spectral function $A(\omega)$ and the imaginary part of the real-frequency Green's function $\Im G(\omega)$,

$$A(\omega) = -\frac{\Im G(\omega)}{\pi}. \tag{7.4}$$

From the well-known Kramers-Kronig transformation, the real part of $G(\omega)$ can be determined as well:

$$\Re G(\omega) = -\frac{1}{\pi}\int_{-\infty}^{\infty} \mathrm{d}\omega' \frac{\Im G(\omega)}{\omega - \omega'}. \tag{7.5}$$

In the HIBISCUS component, we offer a Python script to do this job.

Analytical continuation for the self-energy function: Padé approximation

To calculate real physical quantities, such as the optical conductivity, Seebeck coefficient, thermopower, etc., the self-energy function on the real axis is an essential input. With the Padé approximation[?], we can convert the self-energy function from the Matsubara frequency to real frequency axis. We implemented the Padé approximation for $\Sigma(i\omega_n)$ in the HIBISCUS component.

Analytical continuation for the self-energy function: Gaussian polynomial fitting

The calculated results for the self-energy function on the real axis using Padé approximation strongly depend on the numerical accuracy of the original self-energy data. However, the CT-HYB/DMFT calculations yield a Matsubara self-energy function with numerical noise[?]. In this case, the Padé approximation does not work well. To overcome this problem, Haule *et al.*[?] suggested to split the Matsubara self-energy function into a low-frequency part and high-frequency tail. The low-frequency part is fitted by some sort of model functions which depends on whether the system is metallic or insulating, and the high-frequency part is fitted by modified Gaussian polynomials. It was shown that their trick works quite well even when the original self-energy function is noisy, and is superior to the Padé approximation in all cases. Thus, in the HIBISCUS component, we also implemented this algorithm. It has broad applications in the LDA + DMFT calculations[?].

7.3.1 Maximum entropy method: entropy

7.3.2 Stochastic analytical continuation: sac

7.3.3 Analytical continuation for self-energy: swing

7.3.4 toolbox/makechi

7.3.5 toolbox/makedos

7.3.6 toolbox/makekra

7.3.7 toolbox/makescr

7.3.8 toolbox/makesig

7.3.9 toolbox/makestd

7.3.10 toolbox/maketau

7.3.11 toolbox/makeups

7.3.12 script/p_atomic.py

7.3.13 script/p_ctqmc.py

7.3.14 script/p_hfqmc.py

7.3.15 script/clean.py

7.3.16 script/check.py

7.3.17 script/trailing.sh

7.3.18 script/sar.sh

7.4 Parquet component

As discussed in Sec. 10.1.5, the DMFT + Parquet formalism is a post-processing tool of the *i*QIST package, which can be used to obtain the temperature dependence of various interaction-driven instabilities (ferromagnetic, antiferromagnetic, charge-order, superconductivity) of the system, and furthermore valuable spatial correlation information which is missing in the single-site DMFT. In the DMFT

+ Parquet code, one takes the lattice single-particle Green's function $G(P)$ and local irreducible vertex functions $\Gamma(\omega, \omega', \nu)$ as input, and then uses the Bethe-Salpeter equation and Parquet equations to incorporate momentum-dependence into the lattice two-particle correlation and vertex functions. In practical calculations, due to the huge memory requirements for saving the two-particle quantities [$\chi(P, P', Q)$ and $\Gamma(P, P', Q)$ are three-dimension tensor with double precision complex elements] and the instability issues occurring when performing operations on them (contraction, multiplication, inversion, etc.), one can perform one-shot calculations instead of two-particle self-consistent calculations. A scheme with true self-consistent calculation at both the single- and two-particle level, are still under development. Nevertheless, with the two-particle correlation and vertex functions obtained with the DMFT + Parquet code, one can analyze the gap equation [i.e., Eq. (10.18)] in various interaction channels, to study the instabilities of the system due to competition between lattice, spin, orbital and multi-orbital interactions.

Chapter 8

APPLICATION PROGRAMMING INTERFACES

8.1 Fortran binding

8.2 Python binding

The users can not only execute the components of the iQIST software package directly, but also invoke them in their own programs. To achieve this, we provide simple application programming interfaces (APIs) for most of the components in the iQIST software package in the Fortran, C, C++, and Python languages. With these well-defined APIs, one can easily setup, start, and stop the CT-HYB impurity solvers. For example, one can use the following Python script fragment to start the CT-HYB impurity solver:

```
import mpi4py
import pyiqist as iqist
...
iqist.api.init_ctqmc(myid = 0, num_procs = 10)
iqist.api.exec_ctqmc(iter = 20)
iqist.api.stop_ctqmc()
```

When the computations are finished, one can also collect and analyze the calculated results with Python scripts. Using these APIs, the users enjoy more freedom to design and implement very complex computational procedures and to adapt them to their own requirements.

Chapter 9

*i*QIST IN ACTION

You may have learnt a lot about *i*QIST from the previous chapters. In this chapter, we will show you how to do calculations using *i*QIST through many case studies. Before this, we presume that you have successfully installed *i*QIST package in your systems, and the install directory is /opt/iqist, and the directory /opt/iqist/bin has been added to the PATH environment variable.

```
$ export PATH=/opt/iqist/bin:$PATH
```

If you are new to *i*QIST , please follow the tutorials to learn step by step. We believe that you will have become a master of *i*QIST after finishing all the tutorials. If you have been familiar with *i*QIST , there is no need to start from the beginning, you can choose some cases with your intesets to learn directly.

9.1 Basic applications

9.1.1 Hello *i*QIST !

We first use the *A*zalea component to solve the half-filled single band Hubbard model on the Bethe lattice self-consistently. Due to the density-density Coulomb interaction form in this model, we can use the *A*zalea component based on the segment picture. Let's start the journey!

Step 1: create a working directory, for example, iqsit_test maybe a good choice.

```
$ mkdir iqist_test
```

change directory into iqist_test:

```
$ cd iqist_test
```

create a new directory t911:

```
$ mkdir t911
```

and change directory into t911:

```
$ cd iqist_test
```

now, the current directory is iqist_test/t911.

step 2: prepare input file solver.ctqmc.in

In the directory /opt/iqist/tutor/t911, we have prepared the input file solver.ctqmc.in for *A*zalea , please copy it to the current directory.

```
$ cp /opt/iqist/tutor/t911/solver.ctqmc.in .
```

9.1.2 Mott metal-insulator transition

The users can not only execute the components of the *i*QIST software package directly, but also invoke them in their own programs. To achieve this, we provide simple application programming interfaces (APIs) for most of the components in the *i*QIST software package in the Fortran, C, C++, and Python languages. With these well-defined APIs, one can easily setup, start, and stop the CT-HYB impurity solvers. For example, one can use the following Python script fragment to start the CT-HYB impurity solver:

```
import mpi4py
import pyiqist as iqist
...
iqist.api.init_ctqmc(myid = 0, num_procs = 10)
iqist.api.exec_ctqmc(iter = 20)
iqist.api.stop_ctqmc()
```

When the computations are finished, one can also collect and analyze the calculated results with Python scripts. Using these APIs, the users enjoy more freedom to design and implement very complex computational procedures and to adapt them to their own requirements.

9.2 Advanced applications I: Complex systems

9.2.1 General Coulomb interaction

The users can not only execute the components of the iQIST software package directly, but also invoke them in their own programs. To achieve this, we provide simple application programming interfaces (APIs) for most of the components in the iQIST software package in the Fortran, C, C++, and Python languages. With these well-defined APIs, one can easily setup, start, and stop the CT-HYB impurity solvers. For example, one can use the following Python script fragment to start the CT-HYB impurity solver:

```
import mpi4py
import pyiqist as iqist
...
iqist.api.init_ctqmc(myid = 0, num_procs = 10)
iqist.api.exec_ctqmc(iter = 20)
iqist.api.stop_ctqmc()
```

When the computations are finished, one can also collect and analyze the calculated results with Python scripts. Using these APIs, the users enjoy more freedom to design and implement very complex computational procedures and to adapt them to their own requirements.

9.2.2 Spin-orbital coupling

9.2.3 Crystal field splitting

9.2.4 Retarded interaction and dynamical screening effect

9.3 Advanced applications II: Accurate measurement of physical observables

9.3.1 One-shot and self-consistent calculations

The users can not only execute the components of the iQIST software package directly, but also invoke them in their own programs. To achieve this, we provide simple application programming interfaces

(APIs) for most of the components in the *i*QIST software package in the Fortran, C, C++, and Python languages. With these well-defined APIs, one can easily setup, start, and stop the CT-HYB impurity solvers. For example, one can use the following Python script fragment to start the CT-HYB impurity solver:

```
import mpi4py
import pyiqist as iqist
...
iqist.api.init_ctqmc(myid = 0, num_procs = 10)
iqist.api.exec_ctqmc(iter = 20)
iqist.api.stop_ctqmc()
```

When the computations are finished, one can also collect and analyze the calculated results with Python scripts. Using these APIs, the users enjoy more freedom to design and implement very complex computational procedures and to adapt them to their own requirements.

9.3.2 Data binning mode

9.3.3 Imaginary-time Green's function

9.3.4 Matsubara Green's function and self-energy function

9.3.5 Spin-spin correlation function and orbital-orbital correlation function

9.3.6 Two-particle Green's function and vertex function

9.4 Advanced applications III: post-processing procedures

The users can not only execute the components of the *i*QIST software package directly, but also invoke them in their own programs. To achieve this, we provide simple application programming interfaces (APIs) for most of the components in the *i*QIST software package in the Fortran, C, C++, and Python languages. With these well-defined APIs, one can easily setup, start, and stop the CT-HYB impurity solvers. For example, one can use the following Python script fragment to start the CT-HYB impurity solver:

```
import mpi4py
```

```
import pyiqist as iqist
...
iqist.api.init_ctqmc(myid = 0, num_procs = 10)
iqist.api.exec_ctqmc(iter = 20)
iqist.api.stop_ctqmc()
```

When the computations are finished, one can also collect and analyze the calculated results with Python scripts. Using these APIs, the users enjoy more freedom to design and implement very complex computational procedures and to adapt them to their own requirements.

9.4.1 Analytical continuation for imaginary-time Green's function

9.4.2 Analytical continuation for Matsubara self-energy function

9.5 Practical exercises

9.5.1 Orbital-selective Mott transition in two-band Hubbard model

9.5.2 Orbital Kondo and spin Kondo effects in three-band Anderson impurity model

The users can not only execute the components of the iQIST software package directly, but also invoke them in their own programs. To achieve this, we provide simple application programming interfaces (APIs) for most of the components in the iQIST software package in the Fortran, C, C++, and Python languages. With these well-defined APIs, one can easily setup, start, and stop the CT-HYB impurity solvers. For example, one can use the following Python script fragment to start the CT-HYB impurity solver:

```
import mpi4py
import pyiqist as iqist
...
iqist.api.init_ctqmc(myid = 0, num_procs = 10)
iqist.api.exec_ctqmc(iter = 20)
iqist.api.stop_ctqmc()
```

When the computations are finished, one can also collect and analyze the calculated results with Python scripts. Using these APIs, the users enjoy more freedom to design and implement very complex compu-

tational procedures and to adapt them to their own requirements.

9.6 Library mode

9.6.1 Call iQIST from Fortran language

9.6.2 Call iQIST from Python language

Chapter 10

INSIDE *i*QIST

10.1 Basic theory and methods

In this section, we will present the basic principles of CT-QMC impurity solvers, with an emphasis on the hybridization expansion technique. For detailed derivations and explanations, please refer to Ref.[?] .

10.1.1 Quantum impurity model

The multi-orbital Anderson impurity model (AIM) can be written as $H_{\text{imp}} = H_{\text{loc}} + H_{\text{bath}} + H_{\text{hyb}}$, where

$$H_{\text{loc}} = \sum_{\alpha\beta} E_{\alpha\beta} d^\dagger_\alpha d_\beta + \sum_{\alpha\beta\gamma\delta} U_{\alpha\beta\gamma\delta} d^\dagger_\alpha d^\dagger_\beta d_\gamma d_\delta, \tag{10.1a}$$

$$H_{\text{hyb}} = \sum_{\mathbf{k}\alpha\beta} V^{\alpha\beta}_{\mathbf{k}} c^\dagger_{\mathbf{k}\alpha} d_\beta + h.c., \tag{10.1b}$$

$$H_{\text{bath}} = \sum_{\mathbf{k}\alpha} \epsilon_{\mathbf{k}\alpha} c^\dagger_{\mathbf{k}\alpha} c_{\mathbf{k}\alpha}. \tag{10.1c}$$

In Eq. (10.1), Greek letters in the subscripts denote a combined spin-orbital index, the operator $d^\dagger_\alpha$ (d_α) is creating (annihilating) an electron with index α on the impurity site, while $c^\dagger_{\mathbf{k}\alpha}$ ($c_{\mathbf{k}\alpha}$) is the creation (annihilation) operator for conduction band (bath) electron with spin-orbital index α and momentum $\mathbf{k}$. The first term in H_{loc} is the general form of the impurity single particle term with level splitting and inter-orbital hybridization. This term can be generated by crystal field (CF) splitting or spin-orbit coupling

(SOC), etc. The second term in H_{loc} is the Coulomb interaction term which can be parameterized by intra(inter)-band Coulomb interactions U (U') and Hund's rule coupling J or Slater integral parameters F^k. The hybridization term H_{hyb} describes the process of electrons hopping from the impurity site to the environment and back. H_{bath} describes the non-interacting bath. This Anderson impurity model is solved self-consistently in the DMFT calculations.

10.1.2 Principles of continuous-time quantum Monte Carlo algorithm

We first split the full Hamiltonian H_{imp} into two separate parts, $H_{\text{imp}} = H_1 + H_2$, then treat H_2 as a perturbation term, and expand the partition function $\mathcal{Z} = \text{Tr}e^{-\beta H}$ in powers of H_2,

$$\mathcal{Z} = \sum_{n=0}^{\infty} \int_0^\beta \cdots \int_{\tau_{n-1}}^\beta \omega(\mathcal{C}_n), \tag{10.2}$$

with

$$\omega(\mathcal{C}_n) = d\tau_1 \cdots d\tau_n \text{Tr}\left\{e^{-\beta H_1}[-H_2(\tau_n)]\cdots[-H_2(\tau_1)]\right\}, \tag{10.3}$$

where $H_2(\tau)$ is defined in the interaction picture with $H_2(\tau) = e^{\tau H_1} H_2 e^{-\tau H_1}$. Each term in Eq. (10.2) can be regarded as a diagram or configuration (labelled by $\mathcal{C}$), and $\omega(\mathcal{C}_n)$ is the diagrammatic weight of a specific order-n configuration. Next we use a stochastic Monte Carlo algorithm to sample the terms of this series. In the CT-INT and CT-AUX impurity solvers[??], the interaction term is the perturbation term, namely, $H_2 = H_{\text{int}}$, while $H_2 = H_{\text{hyb}}$ is chosen for the CT-HYB impurity solver[?]. In the intermediate and strong interaction region, CT-HYB is much more efficient than CT-INT and CT-AUX. This is also the main reason that we only implemented the CT-HYB impurity solvers in the *i*QIST software package.

10.1.3 Hybridization expansion

In the hybridization expansion algorithm, due to fact that H_1 does not mix the impurity and bath states, the trace in Eq. (10.3) can be written as $\text{Tr} = \text{Tr}_d\text{Tr}_c$. As a result, we can split the weight of each configuration as

$$\omega(\mathcal{C}_n) = \omega_d(\mathcal{C}_n)\omega_c(\mathcal{C}_n)\prod_{i=1}^{n} d\tau_i. \tag{10.4}$$

$\omega_d(\mathcal{C}_n)$ is the trace over impurity operators (Tr_d), $\omega_c(\mathcal{C}_n)$ is the trace over bath operators (Tr_c). Further, since the Wick's theorem is applicable for the $\omega_c(\mathcal{C}_n)$ part, we can represent it as a determinant of a matrix $\mathcal{Z}_{\text{bath}}\mathcal{M}^{-1}$ with $\mathcal{Z}_{\text{bath}} = \text{Tr}_c e^{-\beta H_{\text{bath}}}$ and $(\mathcal{M}^{-1})_{ij} = \Delta(\tau_i - \tau_j)$. The $\omega_d(\mathcal{C}_n)$ part can be expressed using segment representation when $[n_\alpha, H_{\text{loc}}] = 0$[?]. However, if this condition is not fulfilled, we have

to calculate the trace explicitly, which is called the general matrix algorithm[? ?] . The explicit calculation of the trace for a large multi-orbital AIM with general interactions is computationally expensive. Many tricks and strategies have been implemented in the *i*QIST software package to address this challenge. Please refer to Sec. 10.2 for more details.

In this package, we used importance sampling and the Metropolis algorithm to evaluate Eq. (10.2). The following four local update procedures, with which the ergodicity of Monte Carlo algorithm is guaranteed, are used to generate the Markov chain:

- Insert a pair of creation and annihilation operators in the time interval $[0, \beta)$.
- Remove a pair of creation and annihilation operators from the current configuration.
- Select a creation operator randomly and shift its position on the imaginary time axis.
- Select a annihilation operator randomly and shift its position on the imaginary time axis.

In the Monte Carlo simulations, sometimes the system can be trapped by some (for example symmetry-broken) state. In order to avoid unphysical trapping, we also consider the following two global updates:

- Swap the operators of randomly selected spin up and spin down flavors.
- Swap the creation and annihilation operators globally.

10.1.4 Physical observables

Many physical observables are measured in our CT-HYB impurity solvers. Here we provide a list of them.

Single-particle Green's function $G(\tau)$

The most important observable is the single-particle Green's function $G(\tau)$, which is measured using the elements of the matrix $\mathcal{M}$,

$$G(\tau) = \left\langle \frac{1}{\beta} \sum_{ij} \delta^{-}(\tau, \tau_i - \tau_j) \mathcal{M}_{ji} \right\rangle, \tag{10.5}$$

with

$$\delta^{-}(\tau, \tau') = \begin{cases} \delta(\tau - \tau'), & \tau' > 0, \\ -\delta(\tau - \tau' + \beta), & \tau' < 0. \end{cases} \tag{10.6}$$

Note that in the *i*QIST software package, the Matsubara Green's function $G(i\omega_n)$ is also measured directly, instead of being calculated from $G(\tau)$ using Fourier transformation.

Two-particle correlation function $\chi_{\alpha\beta}(\tau_a, \tau_b, \tau_c, \tau_d)$

The two-particle correlation functions are often used to construct lattice susceptibilities within DMFT and diagrammatic extensions of DMFT. However, the measurements of two-particle correlation functions are a nontrivial task[?] as it is very time-consuming to obtain good quality data, and most of the previous publications in this field are restricted to measurements of two-particle correlation functions in one-band models. Thanks to the development of efficient CT-HYB algorithms, the calculation of two-particle correlation functions for multi-orbital impurity models now become affordable[? ? ?] . In the *i*QIST software package, we implemented the measurement for the two-particle correlation function $\chi_{\alpha\beta}(\tau_a, \tau_b, \tau_c, \tau_d)$, which is defined as follows:

$$\chi_{\alpha\beta}(\tau_a, \tau_b, \tau_c, \tau_d) = \langle c_\alpha(\tau_a) c^\dagger_\alpha(\tau_b) c_\beta(\tau_c) c^\dagger_\beta(\tau_d) \rangle. \tag{10.7}$$

Due to the memory restrictions, the actual measurement is performed in the frequency space, for which we use the following definition of the Fourier transform:

$$\begin{aligned}\chi_{\alpha\beta}(\omega, \omega', \nu) = \frac{1}{\beta} \int_0^\beta d\tau_a \int_0^\beta d\tau_b \int_0^\beta d\tau_c \int_0^\beta d\tau_d \\ \times \chi_{\alpha\beta}(\tau_a, \tau_b, \tau_c, \tau_d) e^{i(\omega+\nu)\tau_a} e^{-i\omega\tau_b} e^{-i\omega'\tau_c} e^{-i(\omega'+\nu)\tau_d}.\end{aligned} \tag{10.8}$$

where ω and ω' [$\equiv (2n+1)\pi\beta$] are fermionic frequencies, and ν is bosonic ($\equiv 2n\pi/\beta$).

Local irreducible vertex functions $\Gamma_{\alpha\beta}(\omega, \omega', \nu)$

From the two-particle Green's function $\chi_{\alpha\beta}(\omega, \omega', \nu)$, the local irreducible vertex function $\Gamma_{\alpha\beta}(\omega, \omega', \nu)$ can be calculated easily, via the Bethe-Salpeter equation[? ? ?] :

$$\Gamma_{\alpha\beta}(\omega, \omega', \nu) = \frac{\chi_{\alpha\beta}(\omega, \omega', \nu) - \beta[G_\alpha(\omega+\nu) G_\beta(\omega') \delta_{\nu,0} - G_\alpha(\omega+\nu) G_\beta(\omega') \delta_{\alpha\beta} \delta_{\omega\omega'}]}{G_\alpha(\omega+\nu) G_\alpha(\omega) G_\beta(\omega') G_\beta(\omega'+\nu)}. \tag{10.9}$$

The $G(i\omega_n)$ and $\Gamma_{\alpha\beta}(\omega, \omega', \nu)$ are essential inputs for the Parquet post-processing code, see Sec. 10.1.5 for more details.

Impurity self-energy function $\Sigma(i\omega_n)$

The self-energy $\Sigma(i\omega_n)$ is calculated using Dyson's equation directly

$$\Sigma(i\omega_n) = G_0^{-1}(i\omega_n) - G^{-1}(i\omega_n), \tag{10.10}$$

or measured using the so-called improved estimator[? ?] . Noted that now the latter approach only works when the segment representation is used. Please check Sec. 10.2.3 for more details.

Histogram of the perturbation expansion order

We record the histogram of the perturbation expansion order k, which can be used to evaluate the kinetic energy [see Eq. (10.15)].

Occupation number and double occupation number

The orbital occupation number $\langle n_\alpha \rangle$ and double occupation number $\langle n_\alpha n_\beta \rangle$ are measured. From them we can calculate for example the charge fluctuation $\sqrt{\langle N^2 \rangle - \langle N \rangle^2}$, where N is the total occupation number:

$$N = \sum_\alpha n_\alpha. \tag{10.11}$$

Spin-spin correlation function

For a system with spin rotational symmetry, the expression for the spin-spin correlation function reads

$$\chi_{ss}(\tau) = \langle S_z(\tau) S_z(0) \rangle, \tag{10.12}$$

where $S_z = n_\uparrow - n_\downarrow$. From it we can calculate the effective magnetic moment:

$$\mu_{\text{eff}} = \int_0^\beta d\tau \chi_{ss}(\tau). \tag{10.13}$$

Orbital-orbital correlation function

The expression for the orbital-orbital correlation function reads

$$\chi^{nn}_{\alpha\beta}(\tau) = \langle n_\alpha(\tau) n_\beta(0) \rangle. \tag{10.14}$$

Kinetic energy

The expression for the system kinetic energy reads

$$E_{\text{kin}} = -\frac{1}{\beta} \langle k \rangle, \tag{10.15}$$

where k is the perturbation expansion order.

Atomic state probability

The expression for the atomic state probability is

$$p_\Gamma = \langle |\Gamma\rangle\langle\Gamma| \rangle, \tag{10.16}$$

where Γ is the atomic state.

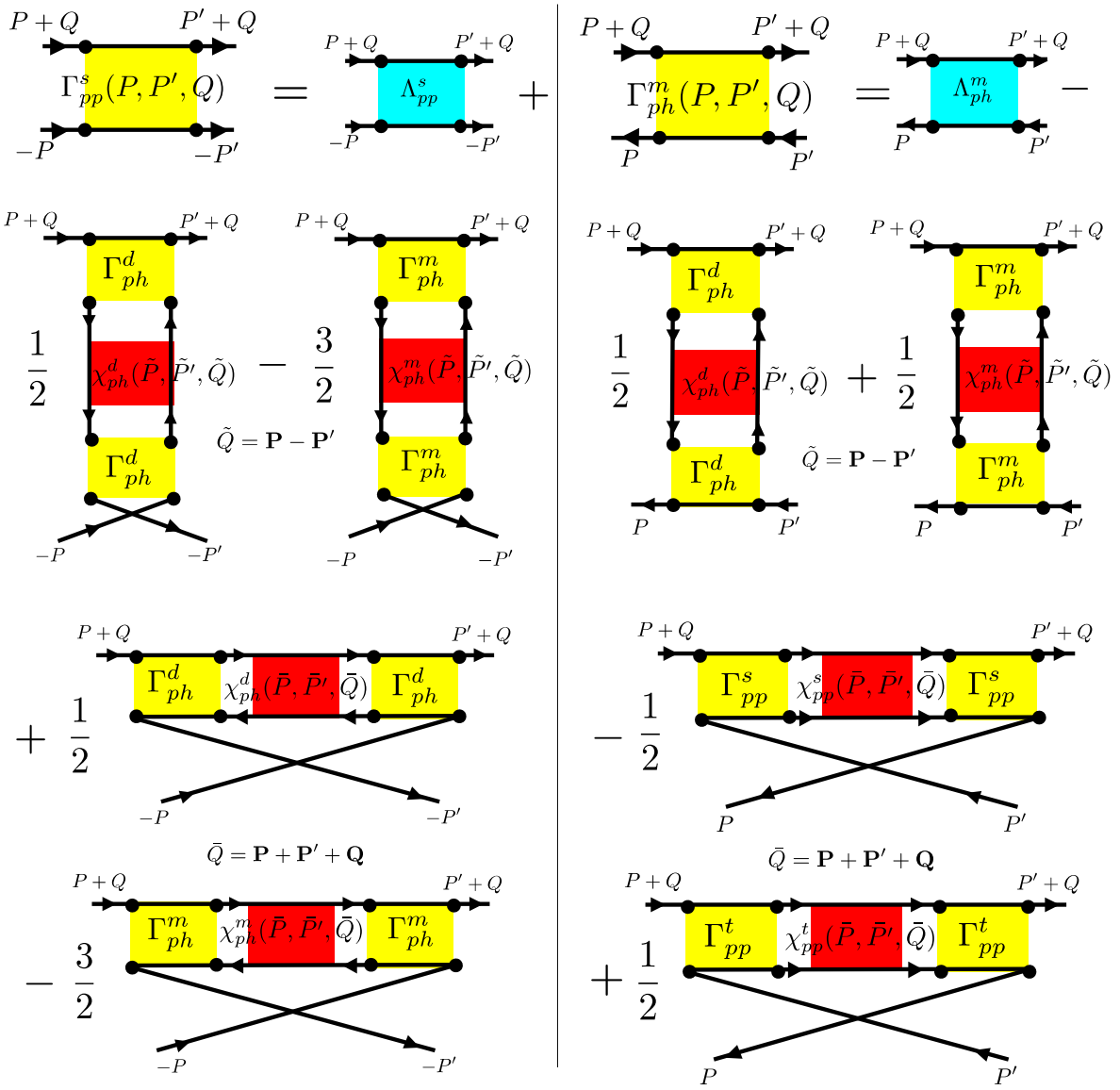

Figure 10.1: (Left panel) Parquet equation of the irreducible vertex function in the pp-s channel, $\Gamma^s_{pp}(P,P',Q)$. It is decomposed into a fully irreducible vertex function Λ^s_{pp} and cross channel contributions from ph-d Φ^d_{ph}, ph-m Φ^m_{ph} vertex ladders. The vertex ladders are products between irreducible vertex functions, Γ, and two-particle correlation functions, χ, in the same channel, with the internal momentum-frequency indices being convoluted, but the momentum-frequency transfer determined by the indices of the irreducible vertex function in the LHS. (Right panel) Parquet equation for the irreducible vertex function in the ph-m channel, $\Gamma^m_{ph}(P,P',Q)$. Here the cross channel contributions also contain pp-s Φ^s_{pp} and pp-t Φ^t_{pp}, vertex ladders.

10.1.5 Two-particle measurements and DMFT + Parquet formalism

CT-HYB/DMFT simulation is a powerful tool to obtain single-particle information. At the two-particle level, however, the correlation function $\chi(\omega, \omega', \nu)$ (which is the generalized susceptibility) and the irreducible vertex functions $\Gamma(\omega, \omega', \nu)$, can only be computed on the impurity site. Hence, the single-site CT-HYB/DMFT technique cannot capture non-local correlation effects of the underlying correlated lattice system. For example, the local particle-particle pairing susceptibilities can provide the information that the system has a superconducting instability in a certain parameter range, but one cannot access the pairing symmetry as the pairing susceptibilities do not have the necessary momentum information. This is an intrinsic limitation of the DMFT method[? ? ?] .

To overcome this difficulty, we develop a new method to go beyond the local nature of DMFT, dubbed DMFT + Parquet formalism[? ?] . We take the local vertex and correlation functions computed in the CT-HYB/DMFT simulations, and bring in the momentum-dependence via two-particle level self-consistent diagrammatic relations -- the Parquet and Bethe-Salpeter equations. The obtained vertex and correlation functions are both momentum- and frequency-dependent. The Parquet equations relate the irreducible vertex function in one interaction channel to those in other channels[? ?] . Usually, we consider four interaction channels: the particle-hole density (ph-d), particle-hole magnetic (ph-m), particle-particle singlet (pp-s), and particle-particle triplet (pp-t) channels[? ? ? ? ? ?] .

In the left (right) panel of Fig. 10.1, the Parquet equation in pp-s (ph-m) channels is shown. One sees that the irreducible vertex function in the pp-s (or ph-m) channel, $\Gamma^s_{pp}(P, P', Q)$ [or $\Gamma^m_{ph}(P, P', Q)$], can be decomposed into a fully irreducible vertex function, $\Lambda^s_{pp}(P, P', Q)$ [or $\Lambda^m_{ph}(P, P', Q)$], and vertex ladders in the ph-d channels [$\Phi^d_{ph}(-P', P+Q, P'-P)$ and $\Phi^d_{ph}(-P', -P, P+P'+Q)$] and ph-m channels [$\Phi^m_{ph}(-P', P+Q, P'-P)$ and $\Phi^m_{ph}(-P', -P, P+P'+Q)$]. Note that in the case of $\Gamma^m_{ph}(P, P', Q)$, the vertex ladders in the ph-d $\Phi^d_{ph}(P, P+Q, P'-P)$, ph-m $\Phi^m_{ph}(P, P+Q, P'-P)$, pp-s $\Psi^s_{pp}(-P-Q, -P, P+P'+Q)$ and pp-t $\Psi^t_{pp}(-P-Q, -P, P+P'+Q)$ channels need to be considered. Here $P \equiv (\mathbf{k}, \omega)$, $P' \equiv (\mathbf{k}', \omega')$, and $Q \equiv (\mathbf{q}, \nu)$, are all momentum-frequency 4-vectors. The rearrangement of the momentum-frequency indices of each vertex ladder is to respect the momentum and energy conservations for the scattering events.

From the lattice Green's function $G(P)$ and local irreducible vertex function $\Gamma(\omega, \omega', \nu)$ obtained in CT-HYB/DMFT, we construct the bubble term $\chi_0(P, Q)$ and then introduce the momentum transfer $\mathbf{q}$ to

the two-particle correlation function, via the Bethe-Salpeter equation (BSE),

$$\chi(\omega,\omega',Q)^{-1}=\left[\frac{1}{N\beta}\sum_{P}\chi_0(P,Q)\right]^{-1}-\Gamma(\omega,\omega',\nu). \tag{10.17}$$

We prepare such $\chi_{ph}^{d/m}(\omega,\omega',Q)$ and $\chi_{pp}^{s/t}(\omega,\omega',Q)$, along with the local irreducible vertex functions $\Gamma_{ph}^{d/m}(\omega,\omega',\nu)$ and $\Gamma_{pp}^{s/t}(\omega,\omega',\nu)$, insert them to the right-hand-side (RHS) of the Parquet equations in Fig. 10.1. Notice that we now have the information of $\tilde{Q}=P'-P$ and $\bar{Q}=P+P'+Q$ in the vertex ladders. Hence, for each fixed Q, we can construct the first order approximated irreducible vertex functions $\Gamma_{ph}^{(1),d/m}(P,P',Q)$ and $\Gamma_{pp}^{(1),s/t}(P,P',Q)$ in the left-hand-side (LHS) of the Parquet equations in Fig. 10.1.

The obtained irreducible vertex functions, $\Gamma_{ph}^{(1),d/m}(P,P',Q)$ and $\Gamma_{pp}^{(1),s/t}(P,P',Q)$, are the first order quantities in a two-particle self-consistent formulation. In the construction of the vertex ladders in Fig. 10.1, we approximate the momentum-frequency convolutions by frequency-only ones, as the input vertex functions are local. To complete the two-particle self-consistent calculation, we then iterate $\Gamma_{ph}^{(1),d/m}(P,P',Q)$ and $\Gamma_{pp}^{(1),s/t}(P,P',Q)$ back to Bethe-Salpeter and Parquet equations to generate the higher order two-particle irreducible vertex and correlation functions, in a successive manner.

With the irreducible vertex functions at hand, we can study the various instabilities of the system generated by the electronic interactions, for example, the superconducting instabilities can be analyzed via the gap equation,

$$\sum_{P'}\Gamma_{pp}^{s/t}(P,P',Q)\chi_0^{pp}(P',Q)\phi(P')=\lambda\phi(P), \tag{10.18}$$

where the leading eigenvalue (LEV) λ and the leading eigenvector $\phi(P)$ reveal the pairing information. Since $\Gamma_{pp}^{s/t}(P,P',Q)\chi_0^{pp}(P',Q)$ is the effective pairing interaction, as temperature approaches the transition temperature T_c, $\lambda\to 1$, and the leading eigenvector $\phi(P)$ reveals the momentum-dependence of the gap function[? ? ?].

10.2 Implementations and optimizations

10.2.1 Development platform

The main part of the *i*QIST software package was developed with the modern Fortran 90 language. We extensively used advanced language features in the Fortran 2003/2008 standard such as an object oriented programming style (polymorphic, inheritance, and module, etc.) to improve the readability and

re-usability of the source codes. The compiler is fixed to the Intel Fortran compiler. We can not guarantee that the *i*QIST can be compiled successfully with other Fortran compilers. Some auxiliary scripts, pre- and post-processing tools are written using the Python language and Bash shell scripts. These scripts and tools act like a glue. They are very flexible and very easily extended or adapted to deal with various problems. In order to avoid incompatibilities, our Python codes only run on the Python 2.x runtime environment.

We use Git as the version control tool, and the source codes are hosted in a remote server. The developers pull the source codes from the server into their local machines, and then try to improve them. Once the development is done, the source codes can be pushed back to the server and merged with the master branch. Then the other developers can access them and use them immediately to start further developments. The members of our developer team can access the private code repository anywhere and anytime.

10.2.2 Orthogonal polynomial representation

Boehnke *et al.*[?] proposed that the Legendre polynomial can be used to improve the measurements of single-particle and two-particle Green's functions. Thanks to the Legendre polynomial representation, the numerical noise and memory space needed to store the Green's function are greatly reduced.

The imaginary time Green's function $G(\tau)$ is expressed using the Legendre polynomial $P_n(x)$ defined in [-1,1]:

$$G(\tau) = \frac{1}{\beta} \sum_{n \leq 0} \sqrt{2n+1} P_n[x(\tau)] G_n, \tag{10.19}$$

here n is the order of Legendre polynomial, G_n is the expansion coefficient, $x(\tau)$ maps $\tau \in [0, \beta]$ to $x \in [-1, 1]$:

$$x(\tau) = \frac{2\tau}{\beta} - 1. \tag{10.20}$$

Using the orthogonal relations of Legendre polynomials, we obtain

$$G_n = \sqrt{2n+1} \int_0^\beta d\tau P_n[x(\tau)] G(\tau). \tag{10.21}$$

If we substitute Eq. (10.5) into Eq. (10.21), we get

$$G_n = -\frac{\sqrt{2n+1}}{\beta} \left\langle \sum_{i=1}^{k} \sum_{j=1}^{k} \mathcal{M}_{ji} \tilde{P}_n(\tau_i^e - \tau_j^s) \right\rangle, \tag{10.22}$$

where

$$\tilde{P}_n(\tau) = \begin{cases} P_n[x(\tau)], & \tau > 0, \\ -P_n[x(\tau+\beta)], & \tau < 0, \end{cases} \tag{10.23}$$

and τ^s and τ^e denote the positions of creation and annihilation operators on the imaginary time axis, respectively. We can also express the Matsubara Green's function $G(i\omega_n)$ using Legendre polynomials:

$$G(i\omega_m) = \sum_{n\leq 0} T_{mn} G_n. \tag{10.24}$$

The transformation matrix T_{mn} is defined as

$$T_{mn} = (-1)^m i^{n+1} \sqrt{2n+1}\, j_n\left[\frac{(2m+1)\pi}{2}\right], \tag{10.25}$$

where $j_n(z)$ is the spheric Bessel function. Actually, in the Monte Carlo simulation, only the expansion coefficients G_n are measured. When the calculation is finished, the final Green's function can be evaluated using Eq. (10.19) and Eq. (10.24). It is worthwhile to note that the T_{mn} do not depend on the inverse temperature β, so that we can calculate and store the matrix elements beforehand to save computer time.

It is easily to extend this formalism to other orthogonal polynomials. For example, in the *i*QIST software package, we not only implemented the Legendre polynomial representation, but also the Chebyshev polynomial representation. In the Chebyshev polynomial representation, the imaginary time Green's function $G(\tau)$ is expanded as follows:

$$G(\tau) = \frac{2}{\beta} \sum_{n\leq 0} U_n[x(\tau)] G_n, \tag{10.26}$$

here the $U_n(x)$ denote the second kind Chebyshev polynomials and $x \in [-1, 1]$. The equation for the expansion coefficients G_n is:

$$G_n = -\frac{2}{\pi\beta} \left\langle \sum_{i=1}^{k} \sum_{j=1}^{k} \mathcal{M}_{ji} \tilde{U}_n(\tau_i^e - \tau_j^s) \sqrt{1 - \tilde{x}(\tau_i^e - \tau_j^s)^2} \right\rangle, \tag{10.27}$$

where

$$\tilde{U}_n(x) = \begin{cases} U_n[x(\tau)], & \tau > 0, \\ -U_n[x(\tau+\beta)], & \tau < 0, \end{cases} \tag{10.28}$$

and

$$\tilde{x}(\tau) = \begin{cases} x(\tau), & \tau > 0, \\ x(\tau+\beta), & \tau < 0. \end{cases} \tag{10.29}$$

Unfortunately, there is no explicit expression for $G(i\omega_n)$ [like Eq. (10.24)] in the Chebyshev polynomial representation.

10.2.3 Improved estimator for the self-energy function

Recently, Hafermann *et al*. proposed efficient measurement procedures for the self-energy and vertex functions within the CT-HYB algorithm[??]. In their method, some higher-order correlation functions (related to the quantities being sought through the equation of motion) are measured. For the case of interactions of density-density type, the segment algorithm is available[?]. Thus, the additional correlators can be obtained essentially at no additional computational cost. When the calculations are completed, the required self-energy function and vertex function can be evaluated analytically.

The improved estimator for the self-energy function can be expressed in the following form:

$$\Sigma_{ab}(i\omega_n) = \frac{1}{2}\sum_{ij} G^{-1}_{ai}(i\omega_n)(U_{jb} + U_{bj})F^{j}_{ib}(i\omega_n), \tag{10.30}$$

where U_{ab} is the Coulomb interaction matrix element. The expression for the new two-particle correlator $F^{j}_{ab}(\tau - \tau')$ reads

$$F^{j}_{ab}(\tau - \tau') = -\langle \mathcal{T} d_a(\tau) d^{\dagger}_b(\tau') n_j(\tau') \rangle, \tag{10.31}$$

and $F^{j}_{ab}(i\omega_n)$ is its Fourier transform. The actual measurement formula is

$$F^{j}_{ab}(\tau - \tau') = -\frac{1}{\beta}\left\langle \sum_{\alpha\beta=1}^{k} \mathcal{M}_{\beta\alpha}\delta^{-}(\tau - \tau', \tau^{e}_{\alpha} - \tau^{s}_{\beta}) n_j(\tau^{s}_{\beta})\delta_{a,\alpha}\delta_{b,\beta} \right\rangle. \tag{10.32}$$

As one can see, this equation looks quite similar to Eq. (10.5). Thus we use the same method to measure $F^{j}_{ab}(\tau - \tau')$ and finally get the self-energy function via Eq. (10.30). Here, the matrix element $n_j(\tau^{s}_{\beta})$ (one or zero) denotes whether or not flavor j is occupied (whether or not a segment is present) at time τ^{s}_{β}.

This method can be combined with the orthogonal polynomial representation[?] as introduced in the previous subsection to suppress fluctuations and filter out the Monte Carlo noise. Using this technique, we can obtain the self-energy and vertex functions with unprecedented accuracy, which leads to an enhanced stability in the analytical continuations of those quantities[?]. In the *i*QIST software package, we only implemented the improved estimator for the self-energy function. Note that when the interaction matrix is frequency-dependent, Eq. (10.30) should be modified slightly[?].

10.2.4 Random number generators

Fast, reliable, and long period pseudo-random number generators are a key factor for any Monte Carlo simulations. Currently, the most popular random number generator is the Mersenne Twister which was

developed in 1998 by Matsumoto and Nishimura[?] . Its name derives from the fact that its period length is chosen to be a Mersenne prime. In the *i*QIST software package, we implemented the commonly used version of Mersenne Twister, MT19937. It has a very long period of $2^{19937}-1$. Of course, if we choose different parameter sets, its period can be shorter or longer.

The Mersenne Twister is a bit slow by today's standards. So in 2006, a variant of Mersenne Twister, the SIMD-oriented Fast Mersenne Twister (SFMT) was introduced[?] . It was designed to be fast when it runs on 128-bit SIMD. It is almost twice as fast as the original Mersenne Twister and has better statistics properties. We also implemented it in the *i*QIST software package, and use it as the default random number generator.

The *i*QIST software package offers other excellent generators as well, such as WELL (Well Equidistributed Long-period Linear)[?] and Marsaglia's XORSHIFT generators[?] ,etc. The XORSHIFT generator is the fastest.

10.2.5 Subspaces and symmetry

For a local Hamiltonian $H_{\rm loc}$ with general interactions, the evaluation of local trace is heavily time-consuming,

$$\omega_d(\mathcal{C}) = {\rm Tr}_{\rm loc}(T_{2k+1}F_{2k}T_{2k}\cdots F_2T_2F_1T_1), \tag{10.33}$$

where $T=e^{-\tau H_{\rm loc}}$ is time evolution operator, F is a fermion creation or annihilation operator, and k is the expansion order for the current diagrammatic configuration $\mathcal{C}$. The straightforward method to evaluate this trace is to insert the complete eigenstates $\{\Gamma\}$ of $H_{\rm loc}$ into the RHS of Eq. (10.33), then

$$\begin{aligned}{\rm Tr}_{\rm loc} \quad = \quad & \sum_{\{\Gamma_1\Gamma_2\cdots\Gamma_{2k}\}} \langle\Gamma_1|T_{2k+1}|\Gamma_1\rangle\langle\Gamma_1|F_{2k}|\Gamma_{2k}\rangle\langle\Gamma_{2k}|T_{2k}|\Gamma_{2k}\rangle\cdots \\ & \times\langle\Gamma_3|F_2|\Gamma_2\rangle\langle\Gamma_2|T_2|\Gamma_2\rangle\langle\Gamma_2|F_1|\Gamma_1\rangle\langle\Gamma_1|T_1|\Gamma_1\rangle. \end{aligned} \tag{10.34}$$

Thus, we must do $4k+1$ matrix-matrix multiplications with the dimension of the Hilbert space of $H_{\rm loc}$. This method is robust but very slow for large multi-orbital impurity model as the dimension of the matrix is impractically large for 5- and 7-band systems, and the expansion order k is large as well.

Actually, the matrices of the fermion operators (F-matrix) are very sparse due to the symmetry of $H_{\rm loc}$. We can take advantage of this to speed up the matrix-matrix multiplications. We consider the symmetry of $H_{\rm loc}$ to find some good quantum numbers (GQNs) and divide the full Hilbert space of $H_{\rm loc}$ with very large dimension into much smaller subspaces labeled by these GQNs[?] . We call a subspace $|\alpha\rangle$

as a superstate? which consists of all the n_α eigenstates of this subspace, $|\alpha\rangle = \{\Gamma_1, \Gamma_2, \cdots, \Gamma_{n_\alpha}\}$. The F-matrix element can only be nonzero between pairs of superstates with different values of GQNs. One fermion operator may bring one initial superstate $|\alpha\rangle$ to some other final superstates $|\beta\rangle$,

$$F|\alpha\rangle = |\beta\rangle, \tag{10.35}$$

or outside of the full Hilbert space. We have to carefully choose some GQNs to make sure that for a fixed initial superstate $|\alpha\rangle$ and a fixed fermion operator, there is one and only one final superstate $|\beta\rangle$ if it doesn't go outside of the full Hilbert space. Given an arbitrary diagrammatic configuration, starting with a superstate $|\alpha_1\rangle$, there will be only one possible evolution path. That is,

$$|\alpha_1\rangle \xrightarrow{F_1} |\alpha_2\rangle \xrightarrow{F_2} |\alpha_3\rangle \xrightarrow{F_3} |\alpha_4\rangle \cdots |\alpha_{2k-1}\rangle \xrightarrow{F_{2k-1}} |\alpha_{2k}\rangle \xrightarrow{F_{2k}} |\alpha_1\rangle. \tag{10.36}$$

The path may break at some point because it goes outside of the full Hilbert space or violates the Pauli principle. For a successful path starting with $|\alpha_1\rangle$, its contribution to the local trace is

$$\begin{aligned} \mathrm{Tr}_{\alpha_1} \quad = \quad & \sum_{\{\Gamma_{\alpha_1}, \Gamma_{\alpha_2}, \cdots, \Gamma_{\alpha_{2k}}\}} \langle \Gamma_{\alpha_1}|T_{2k+1}|\Gamma_{\alpha_1}\rangle \langle \Gamma_{\alpha_1}|F_{2k}|\Gamma_{\alpha_{2k}}\rangle \langle \Gamma_{\alpha_{2k}}|T_{2k}|\Gamma_{\alpha_{2k}}\rangle \cdots \\ & \times \langle \Gamma_{\alpha_3}|F_2|\Gamma_{\alpha_2}\rangle \langle \Gamma_{\alpha_2}|T_2|\Gamma_{\alpha_2}\rangle \langle \Gamma_{\alpha_2}|F_1|\Gamma_{\alpha_1}\rangle \langle \Gamma_{\alpha_1}|T_1|\Gamma_{\alpha_1}\rangle, \end{aligned} \tag{10.37}$$

where $\{\Gamma_{\alpha_i}\}$ are the eigenstates of subspace α_i. Thus, the final local trace should be

$$\mathrm{Tr}_{\mathrm{loc}} = \sum_i \mathrm{Tr}_{\alpha_i}. \tag{10.38}$$

As a result, the original $4k+1$ matrix-matrix multiplications with large dimension reduces to several $4k+1$ matrix-matrix multiplications with much smaller dimensions, resulting in a huge speedup.

In our codes, we implemented several GQNs schemes for different types of local Hamiltonians H_{loc}, which is summarized in Table 10.1. For H_{loc} without spin-orbit coupling (SOC), we have two choices: (1) with Slater parameterized Coulomb interaction matrix, we use the total occupation number N, the z component of total spin S_z as GQNs; (2) with Kanamori parameterized Coulomb interaction matrix, besides N and S_z, we can use another powerful GQN, the so-called PS number? . It is defined as,

$$\mathrm{PS} = \sum_{\alpha=1}^{N_{\mathrm{orb}}} (n_{\alpha\uparrow} - n_{\alpha\downarrow})^2 \times 2^\alpha, \tag{10.39}$$

where α is the orbital index, $\{\uparrow, \downarrow\}$ is spin index, $n_{\alpha\uparrow}$ and $n_{\alpha\downarrow}$ are the orbital occupancy numbers. The PS number labels the occupation number basis with the same singly occupied orbitals. With its help,

Table 10.1: The GQNs supports for various types of local Hamiltonians $H_{\rm loc}$.

GQNs	Kanamori-U	Slater-U	SOC
N, S_z	Yes	Yes	No
N, S_z, PS	Yes	No	No
N, J_z	Yes	Yes	Yes
N	Yes	Yes	Yes

Table 10.2: The total number of subspaces N, maximum and mean dimension of subspaces for different GQNs schemes and multi-orbital models.

	2-band	3-band	5-band	7-band
GQNs	N/max/mean	N/max/mean	N/max/mean	N/max/mean
N, S_z	9/4/1.78	16/9/4.00	36/100/28.44	64/1225/256.00
N, S_z, PS	14/2/1.14	44/3/1.45	352/10/2.91	2368/35/6.92
N, J_z	-	26/5/2.46	96/37/10.67	246/327/66.60
N	5/6/3.20	7/20/9.14	11/252/93.09	15/3432/1092.27

the dimensions of the subspaces become very small, such that we can treat 5-band Kanamori systems efficiently without any approximations. For $H_{\rm loc}$ with SOC, we can use the total occupancy number N and the z component of total angular momentum J_z as GQNs. We summarize the total number of subspaces, maximum and mean dimension of subspaces for different GQNs schemes and multi-orbital impurity models in Table. 10.2. Obviously, using these GQNs can largely reduce the dimension of the F-matrix, and make accurate DMFT calculations for complex electronic systems (such as d- and f-electron materials) possible.

10.2.6 Truncation approximation

As discussed in Sec. 10.2.5, although we have used GQNs to split the full Hilbert space with very large dimension into blocks with smaller dimensions [for cases such as 7-band systems with GQNs (N, J_z) and 5-band systems with GQN (N)], the dimensions of some blocks are still too large and the numbers of blocks are too much so that it is still very expensive to evaluate the local trace. Haule proposed in Ref.[?] to discard some high-energy states because they are rarely visited. For example, for 7-band system with only 1 electron (like Ce metal), only states with occupancy $N = 0, 1, 2$ will be frequently visited, and states with occupancy $N > 2$ can be truncated completely to reduce the large Hilbert space to a very small one. Of course, this truncation approximation may cause some bias because a frequently visited state may be accessed via an infrequently visited state. Therefore, one should be cautious when adopting the truncation approximation, and for example run some convergence tests.

Currently, we adopted two truncation schemes in our codes. The first scheme relies on the occupation number. We just keep those states whose occupation numbers are close to the nominal valence and skip the other states, as shown in the above example. This scheme is quite robust if the charge fluctuations are small enough, such as in the case of a Mott insulating phase. Another scheme is to dynamically truncate the states with very low probability based on statistics which is recorded during the Monte Carlo sampling. This scheme is not very stable, so one needs to use it with caution.

10.2.7 Lazy trace evaluation

The diagrammatic Monte Carlo sampling algorithm consists of the following steps: (1) Propose an update for the current diagrammatic configuration. (2) Calculate the acceptance probability p according to the Metropolis-Hasting algorithm,

$$p = \min(1, \frac{A'}{A} \left|\frac{\omega_c}{\omega'_c}\right| \left|\frac{\omega_d}{\omega'_d}\right|), \tag{10.40}$$

where, A is the proposal probability for the current update and A' for the inverse update, ω_c and ω'_c are the determinants for the new and old configurations, respectively, and ω_d and ω'_d are the local traces for the new and old configurations, respectively. (3) Generate a random number r. If $p > r$, the proposed update is accepted, otherwise it is rejected. (4) Update the current diagrammatic configuration if the proposed update is accepted. It turns out that for CT-HYB, p is usually low ($1\% \sim 20\%$), especially in the low temperature region. On the other hand, the calculation of p involves a costly local trace evaluation. To

avoid wasting computation time when the acceptance probability is very low, in the subspace algorithm, we implemented the so-called lazy trace evaluation proposed in Ref.[?] .

The basic idea of the lazy trace evaluation is simple. For the proposed Monte Carlo move, we first generate a random number r. Then, instead of calculating the local trace from scratch to determine p, we calculate bounds for $|\mathrm{Tr}_{\mathrm{loc}}|$,

$$|\omega_d| = |\mathrm{Tr}_{\mathrm{loc}}| \leq \sum_i |\mathrm{Tr}_i| \leq \sum_i B_i, \tag{10.41}$$

where $B_i \geq |\mathrm{Tr}_i|$. B_i is a product of some chosen matrix norms of T and F matrices:

$$B_i = C\,\|T_{2k+1}\|\,\|F_{2k}\|\cdots\|F_2\|\,\|T_2\|\,\|F_1\|\,\|T_1\| \geq |\mathrm{Tr}(T_{2k+1}F_{2k}\cdots F_2T_2F_1T_1)|\,, \tag{10.42}$$

where C is a parameter depending on the specific type of matrix norm, and $\|\cdot\|$ denotes a matrix norm. If $\mathrm{Tr}_{i'}$ denotes the exact trace of some subspaces, then we have

$$\left||\mathrm{Tr}_{\mathrm{loc}}| - \sum_{i'} |\mathrm{Tr}_{i'}|\right| \leq \sum_{i\neq i'} B_i. \tag{10.43}$$

Thus, we can determine the upper p_{max} and lower p_{min} bounds of p as

$$\begin{aligned} p_{\mathrm{max}} &= R\left(\sum_{i'} |\mathrm{Tr}_{i'}| + \sum_{i\neq i'} B_i\right), \\ p_{\mathrm{min}} &= R\left(\sum_{i'} |\mathrm{Tr}_{i'}| - \sum_{i\neq i'} B_i\right), \end{aligned} \tag{10.44}$$

where $R = \frac{A'}{A}\left|\frac{\omega_c}{\omega_c'}\right|\left|\frac{1}{\omega_d'}\right|$. If $r > p_{\mathrm{max}}$, we reject this move immediately. If $r < p_{\mathrm{min}}$, we accept the move and calculate the determinant and local trace from scratch. If $p_{\mathrm{min}} < r < p_{\mathrm{max}}$, we refine the bounds by calculating the local trace of one more subspace Tr_i until we can reject or accept the move. The calculation of these bounds involves only simple linear algebra calculations of matrix norms which cost little computation time, and one refining operation involves only one subspace trace evaluation. On average, it saves a lot of computation time, as confirmed by our benchmarks.

10.2.8 Divide-and-conquer and sparse matrix tricks

The Monte Carlo updates, such as inserting (removing) a pair of creation and annihilation operators, usually modify the diagrammatic configuration locally. Based on this fact, we implemented a divide-and-conquer algorithm to speed up the trace evaluation. As illustrated in Fig. 10.2, we divide the imaginary

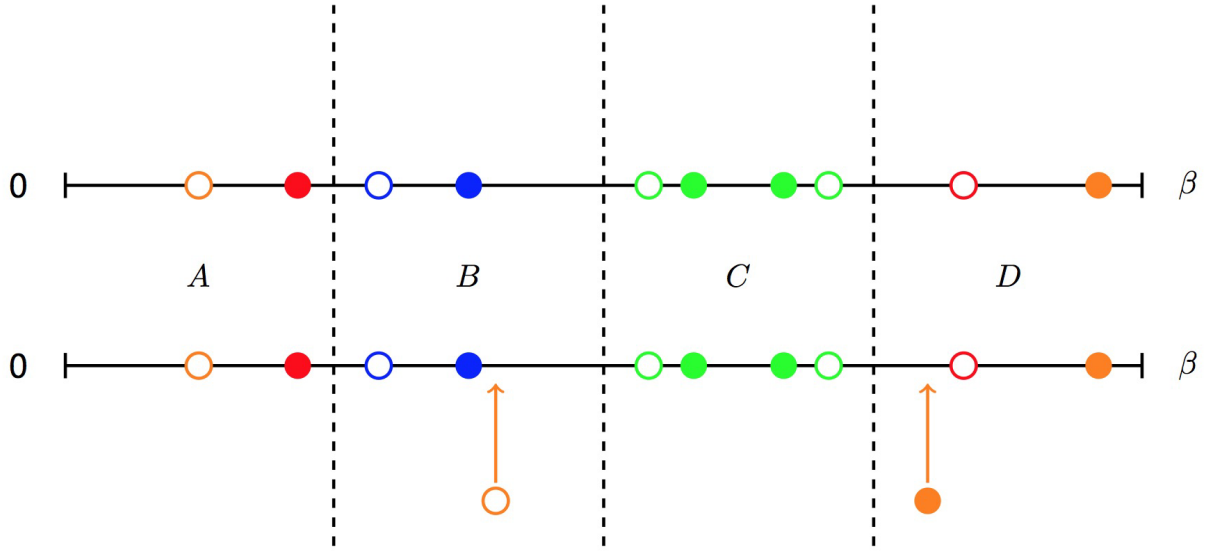

Figure 10.2: Illustration of the divide-and-conquer algorithm. The imaginary time axis is split into four parts with equal length by vertical dashed lines. The open (filled) circles mean creation (annihilation) operators. The color is used to distinguish different flavors. It shows that a creation operator is inserted into the B part, while a annihilation operator is inserted into the D part.

time axis into a few parts with equal length. For each part, there will be zero or nonzero fermion operators, and we save their matrix products when evaluating the local trace in the beginning. In the next Monte Carlo sampling, we first determine which parts may be modified or influenced, and then for these parts we recalculate the matrix products from scratch and save them again. For the unchanged parts, we will leave them unchanged. Finally, we will multiply the contributions of all parts to obtain the final local trace. By using this divide-and-conquer trick, we can avoid redundant computations and speed up the calculation of the acceptance probability p. This trick can be combined with the GQNs algorithm to achieve a further speedup.

If direct matrix-matrix multiplications are used when evaluating the local trace, the F-matrix must be very sparse. Thus, we can convert them into sparse matrices in compressed sparse row (CSR) format, and then the sparse matrix multiplication can be applied to obtain a significant speedup.

10.2.9 Parallelization

All of the CT-HYB impurity solvers in the *i*QIST software package are parallelized by MPI. The strategy is very simple. In the beginning, we launch n processes simultaneously. The master process is responsible for reading input data and configuration parameters, and broadcasts them among the children processes, and then each process will perform Monte Carlo samplings and measure physical observables independently. After all the processes finish their jobs, the master process will collect the measured quantities from all the processes and average them to obtain the final results. Apart from that, no additional inter-process communication is needed. Thus, we can anticipate that the parallel efficiency will be very good, and near linear speedups are possible, as long as the number of thermalization steps is small compared to the total number of Monte Carlo steps.

In practical calculations, we always fix the number of Monte Carlo steps N_{sweep} done by each process, and launch as many processes as possible. Suppose that the number of processes is N_{proc}, then the total number of Monte Carlo samplings should be $N_{\text{proc}}N_{\text{sweep}}$. Naturally, the more processes we use, the more accurate data we can obtain.

10.3 Atomic Solver

The $\mathcal{J}$asmine component of *i*QIST is used to solve a local atomic Hamiltonian defined as,

$$\hat{H}_{\text{atom}} = \hat{H}_0 + \hat{H}_U, \tag{10.45}$$

where, $\hat{H}_0$ is the on-site term, including crystal field (CF) splitting term $\hat{H}_{\text{CF}}$, spin-orbit coupling (SOC) term $\hat{H}_{\text{SOC}}$, and other terms such as a Zeeman term in case of the presence of external magnetic field,

$$\hat{H}_0 = \hat{H}_{\text{CF}} + \hat{H}_{\text{SOC}} + \text{other on-site terms}, \tag{10.46}$$

and $\hat{H}_U$ is the Coulomb interaction term.

We write these terms in second quantization form,

$$\hat{H}_{\text{atom}} = \sum_{\alpha,\beta} E_{\alpha,\beta} \hat{f}^{\dagger}_{\alpha} \hat{f}_{\beta} + \sum_{\alpha,\beta,\gamma,\delta} U_{\alpha,\beta,\gamma,\delta} \hat{f}^{\dagger}_{\alpha} \hat{f}^{\dagger}_{\beta} \hat{f}_{\delta} \hat{f}_{\gamma}, \tag{10.47}$$

where, $\alpha, \beta, \gamma, \delta$ is the single particle orbital-spin index, the first term is the on-site term, the second one is the Coulomb interaction term.

10.3.1 Definition of single particle basis

In this section, we define some single particle basis we used in $\mathcal{J}asmine$ to write down the atomic Hamiltonian $\hat{H}_{\text{atom}}$. We set $\hbar = 1$ in this note.

complex spherical harmonics basis

The complex shperical harmonics $Y_l^m(\theta, \phi)$ is the eigenstate of l^2, l_z,

$$l^2 Y_l^m = l(l+1) Y_l^m, \tag{10.48}$$

$$l_z Y_l^m = m Y_l^m,\ m = -l, -l+1, \cdots, l. \tag{10.49}$$

real spherical harmonics basis

The real spherical harmonics Y_{lm} is defined as

$$Y_{lm} = \begin{cases} \frac{i}{\sqrt{2}} \left(Y_l^{-|m|} - (-1)^m Y_l^{|m|} \right) & if\ m < 0, \\ Y_l^0 & if\ m = 0, \\ \frac{1}{\sqrt{2}} \left(Y_l^{-|m|} + (-1)^m Y_l^{|m|} \right) & if\ m > 0. \end{cases} \tag{10.50}$$

For p system,

$$\begin{aligned} p_x &= Y_{11} = \frac{1}{\sqrt{2}} \left(Y_1^{-1} - Y_1^1 \right), \\ p_y &= Y_{1,-1} = \frac{i}{\sqrt{2}} \left(Y_1^{-1} + Y_1^1 \right), \\ p_z &= Y_{10} = Y_1^0. \end{aligned} \tag{10.51}$$

For d system,

$$\begin{aligned}
d_{z^2} &= Y_{20} = Y_2^0, \\
d_{zx} &= Y_{21} = \frac{1}{\sqrt{2}}\left(Y_2^{-1} - Y_2^1\right), \\
d_{zy} &= Y_{2,-1} = \frac{i}{\sqrt{2}}\left(Y_2^{-1} + Y_2^1\right), \\
d_{x^2-y^2} &= Y_{22} = \frac{1}{\sqrt{2}}\left(Y_2^{-2} + Y_2^2\right), \\
d_{xy} &= Y_{2,-2} = \frac{i}{\sqrt{2}}\left(Y_2^{-2} - Y_2^2\right).
\end{aligned} \tag{10.52}$$

For t_{2g} system ($l \approx -1$), we have a $T - P$ equivalence,

$$\begin{aligned}
d_{zx} &\rightarrow p_y = \frac{i}{\sqrt{2}}\left(Y_1^{-1} + Y_1^1\right), \\
d_{zy} &\rightarrow p_x = \frac{1}{\sqrt{2}}\left(Y_1^{-1} - Y_1^1\right), \\
d_{xy} &\rightarrow p_z = Y_1^0.
\end{aligned} \tag{10.53}$$

For f system,

$$\begin{aligned}
f_{z^3} &= Y_{30} = Y_3^0, \\
f_{xz^2} &= Y_{31} = \frac{1}{\sqrt{2}}\left(Y_3^{-1} - Y_3^1\right), \\
f_{yz^2} &= Y_{3,-1} = \frac{i}{\sqrt{2}}\left(Y_3^{-1} + Y_3^1\right), \\
f_{z(x^2-y^2)} &= Y_{32} = \frac{1}{\sqrt{2}}\left(Y_3^{-2} + Y_3^2\right), \\
f_{xyz} &= Y_{3,-2} = \frac{i}{\sqrt{2}}\left(Y_3^{-2} - Y_3^2\right), \\
f_{x(x^2-3y^2)} &= Y_{33} = \frac{1}{\sqrt{2}}\left(Y_3^{-3} - Y_3^3\right), \\
f_{y(3x^2-y^2)} &= Y_{3,-3} = \frac{i}{\sqrt{2}}\left(Y_3^{-3} + Y_3^3\right).
\end{aligned} \tag{10.54}$$

cubic spherical harmonics basis

The cubic shperical harmonics is defined as the basis of the irreducible representation of cubic point group O_h.

p orbitals:

$$T_{1u} : p_x, p_y, p_z \tag{10.55}$$

d orbitals:

$$\begin{cases} E_g: & d_{z^2}, d_{x^2-y^2} \\ T_{2g}: & d_{zx}, d_{zy}, d_{xy} \end{cases} \tag{10.56}$$

f orbitals:

$$\begin{aligned} f_{x^3} &= -\frac{\sqrt{6}}{4} f_{xz^2} + \frac{\sqrt{10}}{4} f_{x(x^2-3y^2)} \\ f_{y^3} &= -\frac{\sqrt{6}}{4} f_{yz^2} - \frac{\sqrt{10}}{4} f_{y(3x^2-y^2)} \\ f_{z^3} &= f_{z^3} \\ f_{x(y^2-z^2)} &= -\frac{\sqrt{10}}{4} f_{xz^2} - \frac{\sqrt{6}}{4} f_{x(x^2-3y^2)} \\ f_{y(z^2-x^2)} &= \frac{\sqrt{10}}{4} f_{yz^2} - \frac{\sqrt{6}}{4} f_{y(3x^2-y^2)} \\ f_{z(x^2-y^2)} &= f_{z(x^2-y^2)} \\ f_{xyz} &= f_{xyz} \end{aligned} \tag{10.57}$$

$$\begin{cases} T_{1u}: & f_{x^3}, f_{y^3}, f_{z^3} \\ T_{2u}: & f_{x(y^2-z^2)}, f_{y(z^2-x^2)}, f_{z(x^2-y^2)} \\ A_{2u}: & f_{xyz} \end{cases} \tag{10.58}$$

j^2, j_z diagonal basis

Define ϕ_{ljm_j} as the eigenstate of j^2, j_z,

$$j^2 \phi_{ljm_j} = j(j+1)\phi_{ljm_j}, \tag{10.59}$$

$$j_z \phi_{ljm_j} = m_j \phi_{ljm_j}. \tag{10.60}$$

For $j = l + \frac{1}{2}, m_j = m + \frac{1}{2}$,

$$\phi_{ljm_j} = \sqrt{\frac{l+m+1}{2l+1}} Y_l^m \uparrow + \sqrt{\frac{l-m}{2l+1}} Y_l^{m+1} \downarrow . \tag{10.61}$$

For $j = l - \frac{1}{2}, m_j = m + \frac{1}{2}$,

$$\phi_{ljm_j} = -\sqrt{\frac{l-m}{2l+1}} Y_l^m \uparrow + \sqrt{\frac{l+m+1}{2l+1}} Y_l^{m+1} \downarrow . \tag{10.62}$$

Natural basis

The natural basis is defined as the diagonal basis of on-site term $E_{\alpha\beta}$.

10.3.2 Spin-orbit coupling

The spin-orbit coupling (SOC) is implemented at atomic level,

$$\hat{H}_{\mathrm{SOC}} = \lambda \sum_i \vec{\mathbf{l}}_i \cdot \vec{\mathbf{s}}_i, \tag{10.63}$$

where, $\vec{\mathbf{l}}$ is orbital angular momentum, and $\vec{\mathbf{s}}$ is spin angular momentum.

In second quantization form,

$$\hat{H}_{\mathrm{SOC}} = \lambda \sum_{\alpha\sigma,\beta\sigma'} \left\langle \alpha\sigma \left| \vec{\mathbf{l}} \cdot \vec{\mathbf{s}} \right| \beta\sigma' \right\rangle \hat{f}^{\dagger}_{\alpha\sigma} \hat{f}_{\beta\sigma'}, \tag{10.64}$$

where, α is orbital index and σ is spin index, and

$$\vec{\mathbf{l}} \cdot \vec{\mathbf{s}} = \frac{1}{2} \vec{\mathbf{l}} \cdot \vec{\sigma}, \tag{10.65}$$

where, $\vec{\sigma}$ is Pauli operator.

$$\begin{aligned}
\vec{\mathbf{l}} \cdot \vec{\sigma} &= \hat{l}_x \hat{\sigma}_x + \hat{l}_y \hat{\sigma}_y + \hat{l}_z \hat{\sigma}_z \\
&= \begin{bmatrix} 0 & \hat{l}_x \\ \hat{l}_x & 0 \end{bmatrix} + \begin{bmatrix} 0 & -i\hat{l}_y \\ i\hat{l}_y & 0 \end{bmatrix} + \begin{bmatrix} \hat{l}_z & 0 \\ 0 & -\hat{l}_z \end{bmatrix} \\
&= \begin{bmatrix} \hat{l}_z & \hat{l}_x - i\hat{l}_y \\ \hat{l}_x + i\hat{l}_y & -\hat{l}_z \end{bmatrix} \\
&= \begin{bmatrix} \hat{l}_z & \hat{l}_- \\ \hat{l}_+ & -\hat{l}_z \end{bmatrix}
\end{aligned}$$

where, $\hat{l}_{\pm} = \hat{l}_x \pm \hat{l}_y$, and

$$\hat{l}_{\pm} Y_l^m = \sqrt{(l \mp m)(l \pm m + 1)}\, Y_l^{m\pm 1}. \tag{10.66}$$

Write down $\vec{\mathbf{l}} \cdot \vec{\sigma}$ in the complex shperical harmonics basis, the orbital order is:

$$Y_l^{-l} \uparrow, Y_l^{-l} \downarrow, Y_l^{-l+1} \uparrow, Y_l^{-l+1} \downarrow, \cdots, Y_l^{l} \uparrow, Y_l^{l} \downarrow . \tag{10.67}$$

For p system,

$$\vec{\mathbf{l}} \cdot \vec{\sigma} = \begin{bmatrix} -1 & 0 & 0 & \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sqrt{2} \\ \sqrt{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \sqrt{2} & 0 & 0 & -1 \end{bmatrix} \tag{10.68}$$

For t_{2g} system,

$$\vec{\mathbf{l}} \cdot \vec{\sigma} = - \begin{bmatrix} -1 & 0 & 0 & \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sqrt{2} \\ \sqrt{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \sqrt{2} & 0 & 0 & -1 \end{bmatrix} \tag{10.69}$$

For d system,

$$\vec{\mathbf{I}}\cdot\vec{\sigma}=\begin{bmatrix}
-2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & \sqrt{6} & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{6} & 0 & 0 \\
0 & 0 & \sqrt{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 \\
0 & 0 & 0 & 0 & \sqrt{6} & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & -2
\end{bmatrix} \tag{10.70}$$

For f system,

$$\vec{\mathbf{l}}\cdot\vec{\sigma} = \begin{bmatrix}
-3 & 0 & 0 & \sqrt{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -2 & 0 & 0 & \sqrt{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\sqrt{6} & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & \sqrt{12} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \sqrt{10} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{12} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \sqrt{12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \sqrt{10} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \sqrt{12} & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & \sqrt{6} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{10} & 0 & 0 & -2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{6} & 0 & 0 & -3
\end{bmatrix} \tag{10.71}$$

10.3.3 Coulomb interaction

The standard form of Coulomb interaction in second quantization form is:

$$\hat{H}_U = \frac{1}{2}\sum_{\sigma,\sigma'}\sum_{a,b,c,d}\left\langle a\sigma, b\sigma' \left| \frac{1}{r_{12}} \right| c\sigma, d\sigma' \right\rangle \hat{f}^{\dagger}_{a\sigma}\hat{f}^{\dagger}_{b\sigma'}\hat{f}_{d\sigma'}\hat{f}_{c\sigma} \tag{10.72}$$

$$\left\langle a\sigma, b\sigma' \left| \frac{1}{r_{12}} \right| c\sigma, d\sigma' \right\rangle = \int d\vec{r}_1 d\vec{r}_2 \phi^{*}_{\alpha\sigma}(\vec{r}_1)\phi^{*}_{b\sigma'}(\vec{r}_2)\frac{1}{r_{12}}\phi_{c\sigma}(\vec{r}_1)\phi_{d\sigma'}(\vec{r}_2) \tag{10.73}$$

where, $\frac{1}{r_{12}}$ is the Coulomb interaction, $r_{12} = |\vec{r}_1 - \vec{r}_2|$, a, b, c, d is orbital index and $\sigma, \sigma' = \uparrow, \downarrow$ is spin index. In jasmine, we use a array UMAT to store the U tensor, BE VERY CAREFUL WITH THE

ORBITAL ORDER, the indices order of UMAT is the same with that of the Fermion operators, it is NOT the same with that of the U tensor.

Slater Type

Expand $\frac{1}{r_{12}}$ in terms of complex spherical harmonics Y_l^m,

$$\frac{1}{r_{12}} = \sum_k \frac{r_<^k}{r_>^{k+1}} \sum_m (-1)^m C_m^k(\theta_1\phi_1) C_{-m}^k(\theta_2\phi_2) \tag{10.74}$$

where, $C_m^k(\theta\phi) = \sqrt{\frac{4\pi}{2k+1}} Y_l^m(\theta\phi)$.

Set $\phi(\vec{r}) = R_{nl}(r)Y_l^m(\theta\phi)$, then, for fixed n, l, Eqn.10.73 becomes,

$$\begin{aligned}
\left\langle m_1, m_2 \left| \frac{1}{r_{12}} \right| m_1', m_2' \right\rangle &= \sum_{km} (-1)^m c_l^k(m_1, m_1') c_l^k(m_2, m_2') \delta(m + m_1', m_1) \delta(-m + m_2', m_2) F_{nl}^k \\
&= \delta(m_1 + m_2, m_1' + m_2') (-1)^{m_2' - m_2} \sum_k c_l^k(m_1, m_1') c_l^k(m_2, m_2') F_{nl}^k \\
&= \delta(m_1 + m_2, m_1' + m_2') \sum_k c_l^k(m_1, m_1') c_l^k(m_2', m_2) F_{nl}^k
\end{aligned} \tag{10.75}$$

where,

$$c_l^k(m', m'') = \int d\phi d\theta sin(\theta) Y_l^{m'*}(\theta\phi) C_{m'-m''}^k(\theta\phi) Y_l^{m''}(\theta\phi) \tag{10.76}$$

is the Gaunt coefficient for fixed l, and

$$F_{nl}^k = \int_0^\infty r_1^2 dr_1 \int_0^\infty r_2^2 dr_2 R_{nl}^2(r_1) R_{nl}^2(r_2) \frac{r_<^k}{r_>^{k+1}} \tag{10.77}$$

is the Slater integrals.

In this single particle basis, the Coulomb interaction Hamiltonian is:

$$\hat{H}_U = \frac{1}{2} \sum_{m_1, m_2, m_1', m_2', \sigma, \sigma'} \left\langle m_1\sigma, m_2\sigma' \left| \frac{1}{r_{12}} \right| m_1'\sigma, m_2'\sigma' \right\rangle \hat{f}_{m_1\sigma}^\dagger \hat{f}_{m_2\sigma'}^\dagger \hat{f}_{m_2'\sigma'} \hat{f}_{m_1'\sigma} \tag{10.78}$$

where, σ, σ' is spin index.

Set $\alpha = m_1\sigma, \beta = m_2\sigma', \gamma = m_1'\sigma, \delta = m_2'\sigma'$, thus, the UMAT in jasmine is:

$$\textbf{UMAT}(\alpha, \beta, \delta, \gamma) = \frac{1}{2} \delta(m_1 + m_2, m_1' + m_2') \sum_k c_l^k(m_1, m_1') c_l^k(m_2', m_2) F_{nl}^k \tag{10.79}$$

Kanamori Type

The Kanomori type of Coulomb interaction Hamiltonian in jasmine is defined as:

$$\begin{aligned}
\hat{H}_U &= U \sum_a \hat{f}^\dagger_{a,\uparrow} \hat{f}_{a,\uparrow} \hat{f}^\dagger_{a,\downarrow} \hat{f}_{a,\downarrow} \\
&= U' \sum_{a<b,\sigma} \hat{f}^\dagger_{a,\sigma} \hat{f}_{a,\sigma} \hat{f}^\dagger_{b,-\sigma} \hat{f}_{b,-\sigma} \\
&= (U' - J_z) \sum_{a<b,\sigma} \hat{f}^\dagger_{a,\sigma} \hat{f}_{a,\sigma} \hat{f}^\dagger_{b,\sigma} \hat{f}_{b,\sigma} \\
&= -J_s \sum_{a<b,\sigma} \hat{f}^\dagger_{a,\sigma} \hat{f}_{a,-\sigma} \hat{f}^\dagger_{b,-\sigma} \hat{f}_{b,\sigma} \\
&= J_p \sum_{a \neq b} \hat{f}^\dagger_{a,\uparrow} \hat{f}^\dagger_{a,\downarrow} \hat{f}_{b,\downarrow} \hat{f}_{b,\uparrow}
\end{aligned} \tag{10.80}$$

where, a, b is orbital index, and $\sigma = \uparrow, \downarrow$ is spin index.

Appendix

A.1 TODO