ComCTQMC User Guide

Corey Melnick and Patrick Sémon

November 3, 2020

Contents

1 Introduction **2**

2 Theory **2**

3 Observables **4**
3.1 Partition space 5
3.2 Worm space 6

4 Installation and Usage **7**
4.1 Requirements 7
4.2 Installation 7
4.3 Execute 8
4.4 Input and output files 9
4.4.1 Additional I/O formats 10
4.4.2 Input Parameters 11
4.4.3 Output format 17

5 Basis functions **19**

6 Frequently Asked Questions **19**

7 Examples **21**
7.1 Single-Band Hubbard model 21
7.1.1 Input 22
7.1.2 Running the example 26
7.1.3 Results 26
7.2 δ-Plutonium 31
7.2.1 Results 33

1 Introduction

ComCTQMC is a quantum impurity solver which uses the continuous time quantum Monte Carlo (CTQMC) algorithm. In ComCTQMC, the action of the quantum impurty is expanded in terms of the hybridisation function (CT-HYB). It is a stand-alone impurity solver, and it is also embedded in *ab initio* dynamical mean field theory (LDA+DMFT, GW+DMFT) software including ComSuite (github.com/comscopt/comsuite) and Portobello [1].

ComCTQMC features both partition-space and worm-space sampling in order to support the measurement of all one- and two-particle Green's functions along with any static observables which can be extracted from the reduced density matrix.

A GPU accelerated version is available for those with CUDA libraries and CUDA-capable devices (GPUs). GPUs can enable up to 1500x acceleration of f-shell (14 orbital) problems or 5x acceleration of *some* d-shell (10 orbital) problems. (Smaller problems, including higher symmetry d-shell problems should not use GPUs, as they will decelerate the CTQMC.) Acceleration is also notable for cellular DMFT problems with sufficiently large Hilbert spaces.

In this user guide, we will outline the quantum impurity problem solved by ComCTQMC (Sec. 2), list the observables sampled by ComCTQMC (Sec. 3), and describe how to use and install ComCTQMC (Sec. 4). Example runs are provided with the distribution in `ComCTQMC/examples/`, and two of them are described in the last section (Sec. 7).

2 Theory

Let us briefly outline the quantum impurity model solved by ComCTQMC.

An impurity model consists of a small interacting system, the impurity, which hybridizes with baths of non-interacting particles. We consider here hybridization with both a fermionic and a bosonic bath, and the different contributions to the impurity model Hamiltonian, $\hat{H}$, are split into the purely local part, $\hat{H}_{\mathrm{loc}}$; the fermionic and bosonic parts, $\hat{H}_{\mathrm{bath},f}$ and $\hat{H}_{\mathrm{bath},b}$; and the hybridisation between the the baths and the impurity, $\hat{H}_{\mathrm{hyb},f}$ and $\hat{H}_{\mathrm{hyb},b}$. That is,

$$\hat{H} = \hat{H}_{\mathrm{loc}} + \hat{H}_{\mathrm{hyb},f} + \hat{H}_{\mathrm{bath},f} + \hat{H}_{\mathrm{hyb},b} + \hat{H}_{\mathrm{bath},b}, \tag{1}$$

The local part of the impurity Hamiltonian includes both one- and two-body interactions. That is,

$$\hat{H}_{\mathrm{loc}} = \sum_{ij} \hat{c}_i^\dagger (t_{ij} - \mu\delta_{ij}) \hat{c}_j + \frac{1}{2} \sum_{ijkl} \hat{c}_i^\dagger \hat{c}_j^\dagger U_{ijkl} \hat{c}_k \hat{c}_l, \tag{2}$$

where $c_i^\dagger$ creates a fermion in the generalized orbital i, t_{ij} are hopping amplitudes, U_{ijkl} is the interaction tensor, and μ is the chemical potential. The hybridization with the fermionic bath is described by

$$\hat{H}_{\text{hyb},f} = \sum_{i\lambda} \hat{c}_i^\dagger V_{i\lambda} \hat{f}_\lambda + \hat{f}_\lambda^\dagger V_{i\lambda}^* \hat{c}_i \qquad \text{and} \qquad \hat{H}_{\text{bath},f} = \sum_\lambda \epsilon_\lambda \hat{f}_\lambda^\dagger \hat{f}_\lambda, \tag{3}$$

where $\hat{f}_\lambda^\dagger$ creates a fermion in the bath orbital λ with energy ϵ_λ, and the $V_{i\lambda}$ are fermionic hybridization amplitudes. The hybridization with the bosonic bath is described by

$$\hat{H}_{\text{hyb},b} = \sum_{I\kappa} W_{I\kappa} (\hat{b}_\kappa + \hat{b}_\kappa^\dagger) \hat{Q}_I \qquad \text{and} \qquad \hat{H}_{\text{bath},b} = \sum_\kappa \omega_\kappa \hat{b}_\kappa^\dagger \hat{b}_\kappa, \tag{4}$$

where $\hat{b}_\kappa^\dagger$ creates a boson in the bath orbital κ with energy ω_κ, and the $W_{I\kappa}$ are hybridization amplitudes between the bosonic bath and charge degrees of freedom on the impurity,

$$\hat{Q}_I = \sum_{ij} \langle I|ij\rangle \hat{c}_i^\dagger \hat{c}_j. \tag{5}$$

These particle-hole bilinears are Hermitian. That is, $\langle ij|I\rangle = \langle I|ij\rangle^* = \langle I|ij\rangle$, and the hybridization amplitudes $W_{I\kappa}$ are real.

The path integral formalism allows us to integrate out the bath degrees of freedom, and the action of the impurity model can be written as

$$\begin{aligned} S = &- \sum_{ij} \iint_0^\beta \overline{c}_i(\tau) \mathcal{G}_{ij}^{-1}(\tau - \tau') c_i(\tau') d\tau d\tau' \\ &+ \frac{1}{2} \sum_{ijkl} \iint_0^\beta \overline{c}_i(\tau^+) \overline{c}_j(\tau'^+) \mathcal{U}_{ijkl}(\tau - \tau') c_k(\tau') c_l(\tau) d\tau d\tau'. \end{aligned} \tag{6}$$

The the Weiss field

$$\mathcal{G}_{ij}^{-1}(i\nu_n) = (i\nu_n + \mu)\delta_{ij} - t_{ij} - \Delta_{ij}(i\nu_n) \tag{7}$$

absorbs the fermionic bath degrees of freedom, which are encapsulated in the hybridization function

$$\Delta_{ij}(i\nu_n) = \sum_\lambda \frac{V_{i\lambda} V_{j\lambda}^*}{i\nu_n - \epsilon_\lambda}. \tag{8}$$

Similarly, the dynamic interaction

$$\mathcal{U}_{ijkl}(i\omega_n) = U_{ijkl} + \sum_{IJ} \langle kj|I\rangle D_{IJ}(i\omega_n) \langle J|il\rangle \tag{9}$$

absorbs the bosonic bath degrees of freedom, which are encapsulated in the bosonic hybridisation function

$$D_{IJ}(i\omega_n) = \sum_\kappa \frac{2W_{I\kappa}W_{J\kappa}\epsilon_\kappa}{(i\omega_n)^2 - \epsilon_\kappa^2}. \tag{10}$$

Note that we denote the fermionic Matsubara frequencies by $i\nu_n$ and the bosonic Matsubara frequencies by $i\omega_n$.

ComCTQMC can solve most impurities which can be defined within this formalism: It supports complex valued hopping amplitudes t_{ij}, interaction tensors U_{ijkl}, and hybridisation functions (i.e., impurity models where not all $V_{i\lambda}$ can be chosen real, or equivalently, if $\Delta_{ij}(i\nu_n) \neq \Delta_{ji}(i\nu_n)$). Such complex valued impurities may arise when applying a complex valued unitary transformation to the one-particle basis used by default in CTQMC. (See Appendix 5.) However, there are a few restrictions:

1. The particle-hole bilinears $\hat{Q}_I$ in Eq. 5 need to satisfy

   $$[\hat{H}_{\mathrm{loc}}, \hat{Q}_I] = 0 \qquad \text{and} \qquad [\hat{c}_i, \hat{Q}_I] = q_{iI}\hat{c}_i. \tag{11}$$

   Notice that, as a consequence, the $\hat{Q}$'s commute among each other and that the quantum numbers q_{iI} are real. This is because the last equation is equivalent to requiring that $\hat{Q}_I = \sum_i q_{iI}\hat{c}_i^\dagger\hat{c}_i$ and the particle-hole bilinears are Hermitian.

2. This restriction concerns the block-diagonal shape of the hybridization function matrix Δ_{ij} and the local Green's function matrix $G_{\mathrm{loc},ij}(\tau) = -\langle c_i(\tau)\bar{c}_j\rangle_{\mathrm{loc}}$, where $\langle\circ\rangle_{\mathrm{loc}}$ denotes the thermal average with respect to the impurity Hamiltonian $\hat{H}_{\mathrm{loc}}$ in Eq. 2. The requirement is

   $$G_{\mathrm{loc},ij} \equiv 0 \qquad \Rightarrow \qquad \Delta_{ij} \equiv 0. \tag{12}$$

   Equivalently, the non-zero blocks of the hybridization function must lie within the non-zero blocks of the atomic Green function. Since the block-diagonal shape of G_{loc} is usually a consequence of the abelian symmetries of H_{loc}, we may also say that the hybridization is not allowed to break the abelian symmetries of the impurity.

Now, let us discuss the observables which ComCTQMC can sample. (We refer the interested reader to Ref. [2] for a more thorough exploration of the theory behind ComCTQMC.)

3 Observables

ComCTQMC can sample observables in two distinct ways. First, it can measure the observable while in the space of configurations generated by the partition function,

i.e., partition space. Second, it can measure the observable while in the space of configurations generated by the parition function along with some additional operators associated with a local observable (or its improved estimators), i.e., a worm space. (See Ref. [2].)

3.1 Partition space

The following observables are or can be sampled while in partition space.

- Sign: $\langle Z/|Z|\rangle$, The average sign of the sampled configurations across the CTQMC simulation. Each measurement provides approximately “sign" amount of information.
- The expansion histogram: a normalized histogram of the expansion order during the measurement phase of the CTQMC.
- Green’s function: $G^{(1)}_{ij}(\tau) = -\langle T_\tau \hat{c}_i(\tau)\hat{c}^\dagger_j(0)\rangle$.
- Self-energy: From the Dyson equation, $\Sigma = \mathcal{G}^{-1} - G^{-1}$, or the improved estimator.
- Occupations: $\langle c^\dagger_i c_i\rangle$. The average occupation of each orbital is computed by default. The symmetries of the hybridisation matrix are used to simplify the result so that the average occupation of each distinct orbital is output.
- Occupation susceptibility: $\chi_{iiii}(\tau) = -\langle T_\tau \hat{n}^{(ph)}_{ii}(\tau)\hat{n}^{(ph)}_{ii}(0)\rangle$, where $\hat{n}^{(ph)}_{ij} = \hat{c}^\dagger_i\hat{c}_j$.
- Reduced Density Matrix: $(\hat{\rho})_{uv}$ allows one to compute any static observable on the impurity as $\mathrm{Tr}\mathcal{O}(\hat{\rho})_{uv}$. The observables are computed during post-processing by setting options in the input parameter file. A few observable are implemented:
  - Quantum numbers: $\langle q_{n,i} c^\dagger_i c_i\rangle$. ComCTQMC divides the local Hilbert space into a block diagonal form (of sectors). Each of these sectors is defined by a set of quantum numbers. Any quantum number from among these sets can be measured. It is mostly up to the user to know what the good quantum numbers are, but ComCTQMC will check that they are indeed good quantum numbers. (ComCTQMC requires that $q_n|i\rangle_s = q_n|j\rangle_s$ for all quantum numbers q_n, and states $|i\rangle_s$ within sector s, for all sectors of the Hilbert space.) By default only the particle number will be measured.
  - Observables: $\mathcal{O} = \langle t_{ij}c^\dagger_i c_j + v_{ijkl}c^\dagger_i c_j c^\dagger_k c_l\rangle$. ComCTQMC requires that the observables have at least the same symmetries as the local Hamiltonian. That is, if we decompose the Hilbert space of the observable into sectors, those blocks must reside within the blocks of the local Hamiltonian. This is

checked by CTQMC, and any observable which fails to meet the criteria is discarded. By default, no such observable is measured.

The results are reported either by each state in the Hilbert space (**probabilities**) or in total (**scalars**). (The reduced density matrix itself is not output in a human readable format.)

- Quantum-number susceptibility: $\chi_{IJ}(\tau) = \langle Q_I(\tau) Q_J \rangle - \langle Q_I \rangle \langle Q_J \rangle$.

Improved estimators are available for all correlation functions other than the quantum-number susceptibility.

3.2 Worm space

In worm space, one can sample the following observables:

$$G^{(1)}_{ij}(\tau_{12}) = -\langle T_\tau \hat{c}_i(\tau_1) \hat{c}^\dagger_j(\tau_2) \rangle \qquad (*) \tag{13}$$

$$G^{(2)}_{ijkl}(\tau_{12}, \tau_{34}, \tau_{23}) = -\langle T_\tau \hat{c}_i(\tau_1) \hat{c}^\dagger_j(\tau_2) \hat{c}_k(\tau_3) \hat{c}^\dagger_l(\tau_4) \rangle \qquad (*) \tag{14}$$

$$G^{(2,ph)}_{ijkl}(\tau_{12}, \tau_{23}) = -\langle T_\tau \hat{c}_i(\tau_1) \hat{c}^\dagger_j(\tau_2) \hat{n}^{(ph)}_{kl}(\tau_3) \rangle \qquad (*) \tag{15}$$

$$G^{(2,pp)}_{ijkl}(\tau_{12}, \tau_{23}) = -\langle T_\tau \hat{c}_i(\tau_1) \hat{c}_j(\tau_2) \hat{n}^{(pp)}_{kl}(\tau_3) \rangle \qquad (*) \tag{16}$$

$$G^{(2,ph)}_{ijkl}(\tau_{12}) = -\langle T_\tau \hat{n}^{(ph)}_{ij}(\tau_1) \hat{n}^{(ph)}_{lk}(\tau_2) \rangle \tag{17}$$

$$G^{(2,pp)}_{ijkl}(\tau_{12}) = -\langle T_\tau \hat{n}^{(hh)}_{ij}(\tau_1) \hat{n}^{(pp)}_{kl}(\tau_2) \rangle \tag{18}$$

where $\hat{n}^{(ph)}_{ij} = \hat{c}^\dagger_i \hat{c}_j$, $\hat{n}^{(pp)}_{ij} = \hat{c}^\dagger_i \hat{c}^\dagger_j$, $\hat{n}^{(hh)}_{ij} = \hat{c}_i \hat{c}_j$ and $\tau_{12} = \tau_1 - \tau_2$. For the Green functions marked with an asterisk, improved estimators are implemented.

The results are ultimately output as susceptibilities, where the disconnected parts are To do this, the disconnected part of the susceptibility is computed as

$$\chi^{(ph)}_{\mathrm{disc},ijkl}(i\omega_m) = n_i n_k \delta_{ij} \delta_{kl} \delta_{0m} \tag{19}$$

$$\chi^{(pp)}_{\mathrm{disc},ijkl}(i\omega_m) = 0 \tag{20}$$

$$\chi^{(ph)}_{\mathrm{disc},ijkl}(i\nu_n, i\omega_m) = G_{ii}(\nu_n)[n_l \delta_{ij} \delta_{kl} \delta_{0m} - G_{ll}(i\nu_n - i\omega_m)\delta_{ik}\delta_{jl}] \tag{21}$$

$$\chi^{(pp)}_{\mathrm{disc},ijkl}(i\nu_n, i\omega_m) = G_{ii}(i\nu_n) G_{kk}(i\omega_m - i\nu_n)(\delta_{ik}\delta_{jl} - \delta_{il}\delta_{jk}) \tag{22}$$

$$\begin{aligned} \chi_{\mathrm{disc},ijkl}(i\nu_n, i\nu_{n'}, i\omega_m) &= \delta_{ij}\delta_{kl}\delta_{0m} G_{ii}(i\nu_n) G_{kk}(i\nu_{n'}) \\ &\quad - \delta_{il}\delta_{jk}\delta_{nn'} G_{ii}(i\nu_n) G_{kk}(i\nu_n - i\omega_m). \end{aligned} \tag{23}$$

The **self-energy** is output if the one-particle Green's function is measured, and the **full vertex** can be output if the four-point, two-particle Green's function is measured. ($F_{ijkl} = \chi_{ijkl} / G_{ii} G_{jj} G_{kk} G_{ll}$.) The **asymptotic kernels** can be output if all two-particle correlators are measured. (See Ref. [3].)

4 Installation and Usage

In this section, we describe how one can acquire, install, and use ComCTQMC.

4.1 Requirements

- A C++11 capable compiler. The code has been tested using GNU, clang, and intel commpilers. IBM (cray) compilers are not currently supported.
- BLAS and LAPACK libraries – tested with with INTEL MKL, IBM ESSL, and NETLIB-LAPACK libraries.
- (optional) CUDA libraries and compiler – required for the GPU accelerated version of the code. Tested with Cuda/10.1
- (optional) MPI libraries – required for parallelization across CPUs. Tested with OpenMPI, Intel MPI, and (IBM's) Spectrum-MPI libraries.

4.2 Installation

The executables of ComCTQMC are compiled using GNU make. We provide two targets, `cpu` and `gpu`, which compile the un-accelerated and accelerated executables (CTQMC and EVALSIM). Alternatively, one can compile a library, libCTQMC.so, which can be used to embed ComCTQMC in a users code. (The interface is provided by `ComCTQMC/lib/libCTQMC.h`.) Again, two targets are provided, `cpulib` and `gpulib`, which compile the un-accelerated and accelerated library. By default, GNU make will compile the un-accelerated executables. To build any of these targets, simply invoke `make target`. For example,

```
1 make gpu
```

will produce the two GPU accelerated executables in the installation directory (by default, the installation directory is `ComCTQMC/bin/`).

For compilation to succeed, one must first define which compilers and libraries to use, set the compiler flags, and provide paths to the libraries which are not in the system `LD_LIBRARY_PATH`. Set these variables in the file `ComCTQMC/Makefile.in`. There are a number of examples located in `ComCTQMC/Makefile_Examples` which can be used to compile ComCTQMC on Intel, gnu, and clang systems. These examples are commented to explain many of the important flags. Here, let us simply comment on a few of the important options. (For this discussion, we will assume we are building using GNU and OpenMPI.)

- `CXX_MPI`: The MPI enabled compiler. Typically, clusters have wrappers which link the appropriate libraries and set the correct options for that cluster, e.g., mpic++ or mpiicpc. Additionally, ComCTQMC requires that the flag `-DHAVE_MPI` is appended to `CXX_MPI` in order to compile an MPI enabled build.

- `CXX`: The C++ compiler (without an MPI wrapper). Must be supplied to build an MPI and GPU enabled executable or library, so that the CUDA compiler knows which C++ compiler to use.

- `CXX_FLAGS`: The compiler flags to use when compiling the C++ code (but not the CUDA code). Speed optimization, `-O3`, and link-time-optimization, `-flto`, are recommended, although both will substantially lengthen compile time, so they may be turned on only after the user has successfully configured `Makefile.in` for their system.

- `BASE_CPPFLAGS`: Flags for ComCTQMC. The most notable are `-DNDEBUG` which will stop the code from performing a number of safety checks; and `-DUSE_CUDA_SINGLE`, which will compile single-precision GPU kernels, which leads to a massive improvement in problems with particularly large invariant subspaces. While we have not found a problem where double-precision is required, it may be that such problems exist. Therefore, we provide this compiler flag so that users can test the solution when using both single- and double-precision GPU kernels.

- `GPU_ARCH`: Architecture and compute capability of the GPUs. You must set the appropriate architecture flag for your system's GPUs if the GPU accelerated library or executables are desired. (See https://developer.nvidia.com/cuda-gpus#compute)

Finally, one can invoke `make clean` in order to remove the installed components and object files. Use this before recompiling if you change libraries or compiler directives like `-DUSE_CUDA_SINGLE` or optimization flags like `-O3 -flto`.

4.3 Execute

To run ComCTQMC, one must first run the CTQMC code `CTQMC` and then run the post-processing code `EvalSim`. If ComCTQMC is installed in the home directory, one would execute the following commands from the working directory

```
mpirun -n N  ~/ComCTQMC/bin/CTQMC params
mpirun -n M ~/ComCTQMC/bin/EVALSIM params
```

where the control file is named `params.json`. `EVALSIM` can only be parallelized across n^2 CPU's, where n is the number of orbitals. In general, it does not even need to be

parallelized except in very difficult problems, i.e., where the invariant subspaces of the Hilbert space are very large.

4.4 Input and output files

The input and output files are written in the JSON (JavaScript Object Notation) format[1]. Briefly, this format presents a dictionary pairing keys with data. An entry can store integer, real number, string, boolean, array (of those types), or another dictionary. We have added an additional type in order to support complex valued matrices and vectors, which we describe in Sec. 4.4.1. The order of the dictionary entries does not matter. We will refer to a the highest level entry in the JSON dictionary as a block. Let us briefly describe the input files required by ComCTQMC and the output files produced by ComCTQMC before providing a more comprehensive description of the various blocks required in the input and written in the output.

The following input files are required by ComCTQMC:

1. `params.json`: The control file describing the simulation, the quantum impurity, the observables. (See Sec. 4.4.2.)

2. `hybridisation.json`: The hybridisation functions $\Delta_{ij}(i\nu_n)$, labeled according to hybridisation block of the `params.json` file. (See Sec. 4.4.2.) Sufficient Fourier components (Matsubara frequencies) should be included so that the functions are in the asymptotic regime.

3. `dynamical.json` (optional): The dynamical part of the two-body interaction. This is not required if there is no dynamical part.

4. `config_n.json` (restart): The state of the Markov chain `n` from which to resume a simulation. This can be used to continue a simulation or to reduce thermalisation times. (Produced by ComCTQMC.)

5. `params.meas.json` (restart): The state of the estimators at the end of the previous run. Required to continue the simulation and refine the estimators during a restart. (Produced by ComCTQMC.)

During startup, ComCTQMC produces the following files:

1. `defaults.json`: `params.json` with all of the defaults set. Use this, e.g., to learn many of the options which can be set (and are set, by default).

After completion, ComCTQMC produces the following output files:

[1] `https://www.json.org/`

1. `config_n.json`: The state of the Markov chain n at the end of the run.
2. `params.meas.json`: The raw measurements needed to compute the observables.
3. `params.info.json`: A description of the number of steps taken and the number of markov chains simulated across all CPUs.

The post-processing code `EVALSIM` takes these files and produces the following file:

- `params.obs.json`: The observables, e.g., the self-energy, four point vertex functions, susceptibilities, and green's functions. (See Sec. 4.4.3.)

4.4.1 Additional I/O formats

In order to handle complex valued vectors and matrices, ComCTQMC implements two formats: `io::vector<complex>` and `io::matrix<complex>`. First, however, it is good to see an example of the io::vector<real> and `io::matrix<real>` formats which are supported by JSON as an array (or array of arrays) of real numbers. That is,

```
"io::vector<real>": [1.0, 2.0, 3.0],
"io::matrix<real>": [
                [1.0, 2.0, 3.0],
                [4.0, 5.0, 6.0],
                [7.0, 8.0, 9.0]
         ]
```

For the complex versions, we devide the arrays into real and imaginary parts. That is,

```
"io::vector<complex>": {
        "real" : [1.0, 2.0, 3.0],
        "imag" : [1.0, 2.0, 3.0]
        },
"io::matrix<complex>": {
        "real": [
                [1.0, 2.0, 3.0],
                [4.0, 5.0, 6.0],
                [7.0, 8.0, 9.0]
         ],
        "imag": [
                [1.0, 2.0, 3.0],
                [4.0, 5.0, 6.0],
                [7.0, 8.0, 9.0]
         ]
        }
```

In the following sections, we will sometimes distinguish between these formats using the description `io::vector<value>`, where value is either complex (if the parameter “complex" is set to true) or real (if the parameter “complex" is set to false). See Sec. 4.4.2 for a description of the parameter “complex”.

4.4.2 Input Parameters

Here we describe the many input parameters in the `params.json` file. As noted previously, these parameters are organized in blocks within the parameter file. We will mirror that organization here, first describing the higher level purpose of that block and then subsequently listing and describing the individual fields within the block. The data type will be given in parenthesis after the name of the field, which is in bold. The default value will be given after the data type. If there is no default value, we will write none. If there is no default value, the entry is required. If there is a default value, the entry is optional. The data type matrix is used here to denote an array of arrays. The file `defaults.json` contains nearly all options, set to their default value unless overridden by the user in `params.json`.

The miscellaneous parameters which do not belong to their own block are described in Table 1. The block describing the local Hamiltonian is described in Table 2. The block describing the local basis set is described in Table 3. The block describing the hybridisation between the impurity orbitals and bath is described in Table 4. The block describing the measurements which occur in partition space are given in Table 5, while the blocks describing the measurements taken in worm spaces are given in Tables 6 and 7. Finally, Table 8 provides the parameters which describe the asymptotic vertex kernels. (This last block is only used during post-processing).

ComCTQMC will only sample a given worm space if the corresponding parameter block is found in the parameter file. Let us briefly discuss the naming scheme used for the worm observable parameter blocks: *green* refers to the one-particle Green’s function, *susc* to the two-point susceptibilities ,*hedin* to the three-particle susceptibilities, and *vertex* to the four-point susceptibilities. The specifiers *ph* and *pp* refer to the particle-hole and particle-particle channels. The asymptotic kernels can only be computed if all two-particle susceptibilities are measured.

Table 1: Miscellaneous parameters which are not their own block. Required.

complex (boolean, false)	Specify if the impurity is complex or real valued
beta (real, none)	Inverse temperature of the simulation
mu (real, none)	Chemical potential of the impurity.
partition fraction (real, 0.5)	Goal fraction of time a Markov chain should spend in the partition function space. (If no worm spaces are sampled, this setting is ignored.)
sim per device (int, 1)	Number of simulations to run on each available GPU.
thermalisation time (int, none)	Number of minutes to run CTQMC (to reach physical configurations) and Wang-Landau (to determine $\eta_{\mathcal{O}}$) before beginning measurement.
measurement time (int, none)	Number of minutes to continue CTQMC after thermalisation is complete, during which the Markov chains are sampled and observables are measured.
thermalisation steps (int, none)	Number of steps to run CTQMC (to reach physical configurations) and Wang-Landau (to determine $\eta_{\mathcal{O}}$) before beginning measurement. (Overrides thermalisation time.) Note that each simulation (MPI worker and GPU stream) will require (sometimes drastically) different amounts of time to advance a given number of steps, so computational time is wasted when using this option.
measurement time (int, none)	Number of minutes to continue CTQMC after thermalisation is complete, during which the Markov chains are sampled and observables are measured.
measurement steps (int, none)	Number of steps to continue CTQMC after thermalisation is complete, during which the Markov chains are sampled and observables are measured. (Overrides measurement time.) Note that each simulation (MPI worker and GPU stream) will require (sometimes drastically) different amounts of time to advance a given number of steps, so computational time is wasted when using this option.
error (string, none)	Compute error statistics using the Jackknife method. “Serial" or “Parallel” keywords selects how these statics are computed; i.e., across sequential simulationis or across the MPI ranks.
quad insert (bool, false)	Consider four-operator insertion and removal, which can be required for ergodicity in some cases and might help with autocorrelation. Operators are drawn randomly.
seed (real, 41085)	Pseudo random number generator (Mersenne Twister 19937) seed
seed inc (real, 857)	Each Markov chain uses the random seed `(seed)`$+n$`(seed inc)`, where n is the index of the simulation.
expansion histogram (boolean, true)	Accumulate a histogram of the expansion order
trunc dim (integer, none)	Truncates the operator matrices at $i <$`(trunc dim)`. Truncation helps with memory issues and accelerates the simulation, but the simulation is no longer numerically exact. Moreover, the improved estimators may no longer provide the same answer as the non-improved estimators. Use with care.

Table 2: The local Hamiltonian parameter block, “hloc". Required.

one-body (io::matrix<value>, none)	The one-body energy matrix of the impurity, i.e., the t_{ij} from Eq. 2. The matrix is written as an array of arrays, each array having n entries, where n is the number of spin-orbital flavors.
two-body (io::vector[Value], none)	The static two-body interaction matrix, V_{ijkl}, flattened along $i \to j \to k \to l$.
two-body (dictionary)	Instead of entering V_{ijkl} by hand, one can provide a Slater-Condon or Kanamori parameterisation in the `two-body` block. If one uses this block, the local one-particle basis set must also be provided. (See the `basis` block.)
— **parameterisation** (string)	“slater-condon” or “kanamori"
— **U ,J, Uprime** (real, none)	The Kanamori parameters (kanamori). By default, $U' = U - 2J$.
— **F0, F2, F4, ...** (real, none)	The $l + 1$ Slater-Condon parameters (slater-condon)
— **approximation** (string,“none")	“none" or “ising”, where “ising” keeps only the $V_{\alpha\beta\beta\alpha}$ and $V_{\alpha\beta\alpha\beta}$ components, and “none” keeps all components of V.

Table 3: The local one-particle basis parameter block, “basis”, which is required if the two-body interaction matrix is not entered by hand.

type (string, “generic”)	“generic”, “product”, or “coupled”. The underlying basis set. See 5.
orbitals (real, none)	The number of orbitals (generic)
orbitals (string, none)	The orbital momentum: s ($l = 0$), p ($l = 1$), d ($l = 2$), or f ($l = 3$) (product, coupled)
transformation (io::matrix<value>, I)	Transformation matrix which projects or rotates the basis such that $c_i^\dagger = \sum_j U_{ij} \tilde{c}_j^\dagger$.
approximation (string, "none")	Either “ising”, which keeps only the Ising components of the interaction matrix, $V_{\alpha\beta\beta\alpha}$ or $V_{\alpha\beta\alpha\beta}$ or "none", which keeps all components of the interaction matrix.

Table 4: The hybridisation parameter block, “hybridisation’. Required. ’

functions (string, none)	The name of the file containing the hybridisation functions, e.g., “hybridisation.json”. This file contains a json dictionary describing the hybridisation functions $\Delta_{ij}(i\nu_n)$ for $n \geq 0$, where the functions are defined for enough Matsubara frequencies so that the function is in its asymptotic regime. (At which point ComCTQMC can accurately add the high-frequency tail.)
matrix (matrix[string], none)	A matrix of keys which relates the $\Delta_{ij}(i\nu)$ provided in the `hybridisation.json` file to the indices i and j of the associated impurity flavors. If a non-generic basis is used, one must be careful that the hybridisation matrix is ordered according to the enumeration of that basis, as described in 5. If $\Delta_{ij}(i\nu) = 0$, the matrix entry must be the empty string “”. See Sec. 7 for examples.

Table 5: The partition space measurement block, “partition”. Required.

green basis (string, matsubara)	The basis, “matsubara” or “legendre”, in which the green’s function is accumulated.
green bulla (boolean, true)	Use the improved estimator for the green’s function and self-energy.
green matsubara cutoff (real, none)	The largest frequency to measure for the Green’s function, $i\nu_n < 2\pi n/\beta$
green legendre cutoff (real, none)	The number of Legendre components to accumulate. (Required only if measuring in the Legendre basis.)
density matrix precise (boolean, false)	Measure the impurity reduced density matrix in a more accurate, but more expensive manner. Improves static observable estimators and high-frequency tail fits but costs some computational time during sampling. (Not implemented for GPU accelerated CTQMC.)
occupation susceptibility direct (boolean, false)	Susceptibilities for the orbital occupations χ_{iijj} are measured.
occupation susceptibility bulla (boolean, true)	The improved estimators are used to accumulate the χ_{iijj}
quantum numbers (dictionary of array[real])	The quantum numbers of the local Hamiltonian in which the user is interested. Note that *the quantum numbers provided do not affect the CTQMC algorithm.* Instead, they affect which observables are measured during CTQMC (quantum number susceptibilities) or computed during post-processing (static observables and probabilities). They should be written as either arrays, “quantum number name" : $[q_n]$, for $n = 0, 1, .., n_{\text{orbitals}}$. Alternatively, one can supply empty dictionaries for a few common quantum numbers, which directs ComCTQMC to autogenerate the correct array. Currently implemented are N, S_z, and J_z. During post-processing ComCTQMC will check that the supplied quantum numbers are actually quantum numbers of the local Hamiltonian and remove those which are not.
quantum number susceptibility (boolean, false)	The susceptibility for the quantum numbers, χ_I, are accumulated. See Sec. 7 for an example.
susceptibility cutoff (real, none)	The largest frequency to measure for the susceptibilities, $i\omega_n < 2\pi n/\beta$
observables (dictionary of hamiltonians)	The static observables in which the user is interested. These observables must commute with the local Hamiltonian. They are entered in nearly the same way the local Hamiltonian is, i.e., as a one-body matrix and/or two-body vector. (If one of these matrices is not provided, it is assumed to be zero.) The two-body matrix must be set explicitly if it is non-zero, i.e., one cannot use the Slater-Condon or Kanamori parameterizations. This parameter is only used during post-processing, at which point ComCTQMC will check that the supplied observable meets the criterion and remove those which do not.

Table 5: The partition space measurement block **continued**.

probabilities (array[string], empty)	An array listing the names of the quantum number and/or observables for which the probability should be computed during post-processing so that one can generate histograms of the valence occupancy. The results will be given in three fields in the output "probabilities" block: "occupation numbers", "quantum numbers", and "surviving." These provide the probability that each state is occupied (in occupation representation, read from left-to-right); the quantum number, observables, and probability of state; and the list of quantum numbers and observables which meet the criterion required by ComCTQMC (in the same order as the results are listed in the "quantum numbers" section. Adding the keywords "energy" to the list instructs ComCTQMC to also compute and output the energy of each state. Adding "label" instructs Comctqmc to print a unique identifier for each state (so that one can more easily connect the occupation state representation and the quantum number results.
sweepA (real, 50)	Number of steps taken by Markov chain in partition function space before sampling those observables which are computationally inexpensive to sample (Susceptibilities, Green's functions, density matrix).
sweepB (real, 250)	Number of steps before sampling those observables which are more computationally expensive to sample (Improved estimators and precise density matrix).

Table 6: The two-point worm space measurement blocks, "green", "susc ph ", and "susc pp ". Required for worm measurements of these observables.

basis (string, "matsubara")	Basis, "matsubara" or "legendre", in which to accumulate the observable
cutoff (string, none)	Number of Matsubara or Legendre components to measure, $n <$`(cutoff)`.
meas (array[string], ["imprsum"])	Manner, "", or "imprsum", in which to accumulate the observable, i.e., with or without the improved estimator. The improved estimators are not implemented for `susc ph` or `susc pp` worm spaces.
sweep (real, 50)	Number of steps between samples of the worm space.

Table 7: The three- and four-point worm space measurement blocks, “hedin ph ”, “hedin pp ”, and “vertex”. Required for worm measurements of these observables.

basis (string, “matsubara”)	Basis, “matsubara” or “legendre”, in which to accumulate the observable
fermion cutoff (int, none)	Number of fermionic Matsubara or Legendre components to measure, $\|n\| <$`(fermion cutoff)`
boson cutoff (int, none)	Number of fermionic Matsubara or Legendre components to measure, $m <$`(boson cutoff)`.
matsubara cutoff (int, 50)	Number of fermionic Matsubara components to compute during post-processing when measured in the mixed Legendre-Fourier basis
meas (array[string], [“imprsum”])	Manner, “”, or “imprsum”, in which to accumulate the observable, i.e., with or without the improved estimator.
sweep (real, 50)	Number of steps between samples of the worm space.
full (boolean, false)	Evaluate the corresponding full vertex function during post-processing (only implemented for `vertex`)

Table 8: The asymptotic kernel parameter block, “kernels”. Required for computation of kernels.

full (boolean, false)	Compute the full vertex using the asymptotic kernel functions during post-processing.
fermion cutoff (int, none)	Number of fermionic Matsubara components for which to compute the asymptotic vertex, $\|n\| <$`(fermion cutoff)`.
boson cutoff (int, none)	Number of bosonic Matsubara components for which to compute the asymptotic vertex, $m <$`(bosono cutoff)`.
asymptotic cutoff (int, 10)	Condition for replacing the measured vertex with asymptotic vertex, `(asymptotic cutoff)`$^4 \geq \frac{\beta^4}{\pi^4}\nu(\nu-\omega)\nu'(\nu'-\omega)$

4.4.3 Output format

Two human-readable files are produced during a full run of ComCTQMC, i.e., an invocation of `CTQMC` and `EVALSIM`: `params.info.json` and `params.obs.json`. The former is essentially self-explanatory: It lists the number of steps taken across all Markov chains (in both measurement and thermalisation phases), the number of markov chains, and the number of MPI processes which worked on the problem. While the latter is hopefully also self-explanatory for the most part, it does require some additional explanation.

The main output file, `params.obs.json`, is organized into blocks in much the same way the main input file, `params.json`, is organized. That is, there is a block for each partition space sampled, e.g., there will be “partition" and “green imprsum" blocks, provided that the input file contains the blocks ‘partition" and “green” (where “green” contains “meas”: [“imprsum”]). In each of these blocks, there is a list of the observables measured in that space. For example, the “green imprsum” block will have entries for the Green’s function (“green”) and self-energy (“self-energy”). Let us go over the format of the observables:

- Complex valued functions are written in the io::vector<complex> format (Sec. 4.4.1).
- One expects a lot of observables to be real valued, even if “complex” is set to true (and one is therefore simulating a complex valued impurity). For example, the high-frequency moments of the Green’s function or occupations of each orbital are real valued. Still, ComCTQMC represents most quantities using io::vector<value>. The exception to this rule are the static observables associated with “quantum numbers”, “observables”, and “probabilities”; and the simulation descriptors like the the expansion order histogram and sign. These exceptions are always represented as an io::vector<real>.
- Each function in an observable is labeled as follows:
  - One-particle correlators, e.g., G_{ij} and Σ_{ij}, are labeled according to the input hybridisation matrix. That is, index pair i, j corresponds to the entry “A” in the hybridisation matrix, so ComCTQMC will output the function “A” instead of function “ij”. Symmetries in the hybridisation matrix are used to reduce the number of reported functions. For example, if both ij and kl correspond to the entry “A”, then the measured functions G_{ij} and G_{kl} will be averaged and reported as function “A”.
  - Two-particle correlators measured in worm spaces, e.g., χ_{ijkl} and F_{ijkl}, are labeled according to their indices $ijkl$ with a small adjustment: Creation

operators are mapped to the index $i \to 2i$ and annihilation operators are mapped to the index $i \to 2i+1$. For example, χ^{ph}_{1122} is reported as "0_1_3_2" and χ^{pp}_{1221} is reported as "0_2_3_1".

  - Susceptibilities measured in partition spaces, χ_{qn} or χ_{iijj}, are labeled according to the quantum number or the index $i \to 2i$ and $j \to 2j+1$.

- Green's functions, vertices (including the self-energy), and susceptibilities are separated into the measured "function" and computed high-frequency "moments".

- Frequencies are reported in the following order:

  - One-time correlators: $i\nu_n$ or $i\omega_n$ from frequencies in [0,cutoff).
  - Multi-time correlators, e.g., $\chi(i\nu_n, i\nu_{n'}, i\omega_m)$ are reported with the fastest to slowest index of $n \to n' \to m$. Fermionic indices always span both positive and negative values up to the cutoff frequency. Bosonic indices span both positive and negative values for complex impurities and [0,boson cutoff) for real impurities.

- Probabilities are reported in two forms.

  - "occupations": The probability that each state in occupation state representation is occupied. It is given as an array of arrays. Each inner array contains occupation state representation (read from left to right as a list of 0's and 1's), unique, real valued serial number (as a floating point number), and then the probability this state is occupied (as an io::vector<real>).
  - "quantum numbers" provides probabilities that states with a given set of quantum numbers and observables are occupied. It is given as an array of arrays. The inner arrays list the value of the quantum numbers and/or observables which commute with the local Hamiltonian in the order they were entered in the "probabilities" field in the `params.json`. (The "surviving" field in the "probabilities" block will list those quantum numbers and observables which commute.) The final entry in each array is the probability this state is occupied (as an io::vector<real>). One can generate a serial number for these entries which corresponds with the occupation state representation. To do this, add "label" to the "probabilities" list in `params.json`.

We repeat that for the most part we have tried to make this file human readable. Inspection of the file should help with any questions not answered in this section. Now, let us discuss the basis functions which make the simulation of real materials more convenient.

5 Basis functions

In general, one does not need to know the form of the one-particle basis to run ComCTQMC. However, ComCTQMC provides Slater-Condon and Kanamori parameterisations of the static two-body interaction tensor. In order to use these parameterisations, it is necessary to know this basis set. For this purpose, ComCTQMC implements three basis sets. A generic basis set, which is all that is required for the Kanamori interaction; and the product and spin-coupled basis sets, which provide the detail required for the Slater-Condon interaction.

The product basis set is defined here as

$$|l, m, \sigma\rangle = Y_{l,m} \otimes \sigma, \tag{24}$$

where σ is the spin and $Y_{l,m}$ are the real spherical harmonics, which are defined as

$$Y_{\ell,m} := \begin{cases} i\sqrt{2}\left(Y_\ell^m - (-1)^m Y_\ell^{-m}\right) & \text{if } m < 0 \\ Y_\ell^0 & \text{if } m = 0 \\ 1\sqrt{2}\left(Y_\ell^{-m} + (-1)^m Y_\ell^m\right) & \text{if } m > 0 \end{cases} \tag{25}$$

in terms of the complex spherical harmonics Y_l^m.

The coupled basis set is defined here as

$$|j, m_j\rangle = \sum_{m\sigma} Y_l^m \otimes \sigma \langle \ell, m; \frac{1}{2}, \sigma | j, m_j \rangle, \tag{26}$$

where $\langle \ell, m; \frac{1}{2}, \sigma | j, m_j \rangle$ are the Clebsch-Gordan coefficients.

The generic, product, and coupled basis are enumerated as

$$|1, \downarrow\rangle, ..., |n, \downarrow\rangle, |1, \uparrow\rangle, ..., |n, \uparrow\rangle \tag{27}$$

$$|\ell, -\ell, \downarrow\rangle,\ |\ell, -\ell+1, \downarrow\rangle, \ldots,\ |\ell, \ell, \downarrow\rangle,\ |\ell, -\ell, \uparrow\rangle,\ |\ell, -\ell+1, \uparrow\rangle, \ldots,\ |\ell, \ell, \uparrow\rangle \tag{28}$$

$$|\ell - \frac{1}{2}, -\ell + \frac{1}{2}\rangle, \ldots,\ |\ell - \frac{1}{2}, \ell - \frac{1}{2}\rangle,\ |\ell + \frac{1}{2}, -\ell - \frac{1}{2}\rangle, \ldots,\ |\ell + \frac{1}{2}, \ell + \frac{1}{2}\rangle. \tag{29}$$

As discussed in Sec. 4.4.2, the user can apply any unitary transformation to these basis. How does this effect the CTQMC? It transforms the two-body interaction matrix associated with the respective basis. The one-body Hamiltonian, hybridisation matrices, and quantum numbers are **not** transformed.

6 Frequently Asked Questions

Here we describe some common questions and errors, providing answers and solutions:

- **Questions**:

  1. **Can I solve problems with an off-diagonal hybridisation function?**: In general, yes. However the CTQMC algorithm can fail on these problems either by having a very small sign (or in more subtle ways). In particular, we have found that problems in which the hybridisation functions on the off-diagonal are much stronger than those on the diagonal may not be reliably solved by CTQMC. We suggest checking that the improved estimator ("self energy" in the partition block) and non-improved estimator ("self energy dyson' in the same block') provide the same result within error bars. (To perform this check, one should set "green bulla" : true in the partition block of the parameter file.)
  2. **What kinds of dynamical interactions can I include in my impurity problem?**: The dynamical interactions must be built from quantum numbers which commute with the local Hamiltonian. See the error discussed below, *problem with quantum numbers*.

- **Errors**:

  1. **jsx::parser: error while reading object**: One of the json files is not correctly formatted. Often, this means that you are missing a comma after a field or have an extra (or missing) bracket.
  2. **Problem with quantum numbers**: We require that the quantum numbers listed in "quantum numbers" or used to build the dynamical interaction commute with the local Hamiltonian. ComCTQMC will check that they do. If they do not, this error will be thrown. The total number of electrons, `N` or [1,1,1,...,1], is always a good quantum number, but $n_i = [0, 0, ..., 0, 1, 0..., 0, 0]$ rarely is (due to the two-body interaction tensor). In the relativistic case, J_z is a good quantum number in the native basis, but unitary transformations to this basis (according to the symmetries of the crystal) may mean that only J^2 is a good quantum number. If there is no dynamical interaction, the quantum number field only effects the calculation of those observables which are defined by them, e.g., the quantum number susceptibilities. It does not affect the basic CTQMC algorithm. Therefore, if you do not care about these observables, you may safely remove the field from your input file. If, however, you are including a dynamical interaction (which must be built from quantum numbers), then it is crucial use good quantum numbers. The post-processing script `EVALSIM` will tell you which quantum numbers survive the requirement and which don't. So, you can run CTQMC without

any quantum numbers and then test the quantum numbers of interest using `EVALSIM` and the "quantum numbers" and "probabilities" fields.

3. **ut::Zahl<value>: Constructor is not a real number** or **ut::Zahl<value>: division by zero**: Zahl is our big number class. These warnings mean that ComCTQMC has generated a `NaN` or `Inf` (or is about to). Typically this arises because the problem posed to ComCTQMC is not physical. For example, if a hybridisation function grows to infinity as the frequency goes to infinity, this error will arise. We have never encountered this problem in a well posed DMFT or CDMFT problem, but do encounter it when there is a bug in our DMFT or CDMFT code.

7 Examples

In this section, we will present an example showing how to use ComCTQMC codes CTQMC and EVALSIM as well as showcase some results. Further description of the control file and the available options are present in the program documentation bundles with the ComCTQMC archive. Note that 4 explains how to install and use ComCTQMC and describes all input parameters.

7.1 Single-Band Hubbard model

In this example, we will show how to run a simple example: A single-band Hubbard impurity model. The Hamiltonian is

$$H = \sum_{\boldsymbol{k}\sigma} \epsilon_{\boldsymbol{k}} c^\dagger_{\boldsymbol{k}\sigma} c_{\boldsymbol{k}\sigma} + \epsilon_f \sum_\sigma f^\dagger_\sigma f_\sigma + U n_{f,\uparrow} n_{f,\downarrow} + \sum_{\boldsymbol{k}\sigma} V_{\boldsymbol{k}} (c^\dagger_{\boldsymbol{k}\sigma} f_\sigma + f^\dagger_\sigma c_{\boldsymbol{k}\sigma}), \tag{30}$$

where $c^\dagger_{\boldsymbol{k}\sigma}$ and $c_{\boldsymbol{k}\sigma}$ are the annihilation and creation operators for the non-interacting bath state with wavevector $\boldsymbol{k}$, spin σ, and non-interacting energy $\epsilon_{\boldsymbol{k}}$, and $f^\dagger_\sigma$ and f_σ are the annihilation and creation operators for the interacting impurity state with spin σ and energy ϵ_f. Here solve this model exactly for the case where there a three bath states at $\epsilon_\pm \pm D/2$ and $\epsilon_0 = 0$, such that $D = 1$ is the bandwidth and our energy scale, and set $U = 4D$, $V_{\boldsymbol{k}} = V = D$, and $\epsilon_f = -U/2$ (half-filling).

We choose this simple model, and solve it at a moderate temperature $\beta = 10$, for a two reasons: First, its possible to run ComCTQMC on a laptop and replicate these results, particularly on modern laptops which offer O(10) CPU threads. Second, exact diagonalization (ED) [4] can quickly provide the exact results for this model if we model an impurity hybridized with a small, finite number of bath states. In this section, we will use this latter fact in order to compare ComCTQMC with the exact solution and highlight its various capabilities.

Next, we will go over the input files and parameters required to simulate this impurity. Then, we will reiterate how ComCTQMC and is used to solve the impurity problem. Finally, results will be presented, and those results will be compared with the exact results as computed by exact diagonalization.

7.1.1 Input

In ComCTQMC, the impurity model arising from this lattice is defined by the following portion of the input file

```
{
        ...
    "hloc" : {
        "one body" : [
            [0.0, 0.0],
            [0.0, 0.0]],
        "two body": [ 0, 0, 0, 0, 0, 0, 4, 0, 0, 4, 0, 0, 0, 0, 0, 0 ]
        },
    "hybridisation": {
        "functions": "hyb.json",
        "matrix": [
            [ "A", "" ],
            [ "", "A" ] ]
    },
        "beta": 10,
        "mu": 2,
        ...
 }
```

As discussed in Sec. 4.4, these parameters define the following:

- `beta`: The inverse temperature.
- `mu`: The chemical potential.
- `hloc`: The impurity Hamiltonian H_{loc}.
  - `one body`: The one particle tensor t_{ij} in matrix form, c.f., Eq. (7).
  - `two body`: The two body tensor V_{ijkl} flattened along $i \to j \to k \to l$.
- `hybridisation`: The hybridisation function Δ is defined
  - `matrix`: The entries Δ_{ij} of the hybridization function are labeled (empty quotes represent entries which are zero). Non-zero off-diagonal entries (not

shown here) are supported, as long as the hybridization function is block diagonal and all entries in a block are non-zero. It is also sufficient that the hybridization function acquires a block diagonal form after some permutation σ of the orbitals. That is, $(\Delta_{\sigma(i)\sigma(j)})_{ij}$ is block diagonal.

- `functions`: The file name (`hyb.json`) contains the function for each label. In this example, `hyb.json` looks like this:

```
{
    "A" : {
        "imag" : [ -0.1518943897, -0.1260521182, -0.09894884474, -0.07885927836, ... ],
        "real" : [ 0.01183399821, 0.006447214601, 0.005432068729, 0.005127453585, ... ]
    }
}
```

  The functions in are written in Matsubara frequencies ω_n, with an array $[\mathrm{Re}\Delta(i\omega_1), \mathrm{Re}\Delta(i\omega_2), \dots]$ for the `real` part and an array $[\mathrm{Im}\Delta(i\omega_1), \mathrm{Im}\Delta(i\omega_2), \dots]$ for the `imag`-inary part. The positive Matsubara frequencies are sufficient to define the hybridization function since $\Delta_{ij}(-i\omega_n) = \overline{\Delta_{ji}}(i\omega)$ by hermicity of the impurity model Hamiltonian. ***Note***: The tail of the functions is added by the impurity solver, that is, the functions should be entered up to frequencies where the asymptotic behavior $\propto (i\omega_n)^{-1}$ is reached.

Additionally, we need to tell ComCTQMC how long it should run. This is accomplished through the `measurement time`, and `thermalisation time` fields in the input file. For example,

```
{
        ...
    "measurement time" = 59,
    "thermalisation time" = 1
        ...
 }
```

tells ComCTQMC to run for an hour, 59 minutes of which is spent accumulating the estimators specified by the code and 1 minute of which is spent thermalizing the initial configuration and determining the relative volumes of the configuration spaces (if any worm spaces are to be measured).

Finally, we need to tell ComCTQMC which observables to sample. The user is required to have `partition` block, wherein the user specifies which observables to sample in partition space.

```
{
    ...
    "partition": {
        "green matsubara cutoff": 50,
    },
    ...
}
```

While many observables and control flags are available, only `green matsubara cutoff` is required. Here we specify that ComCTQMC should sample all matsubara frequencies $0 < i\omega_n < 50$.

Additionally, one can specify the worm spaces to sample. To sample all observables, we write

```
{
        ...
    "green": {
        "cutoff": 25,
    },
    "susc_ph": {
        "cutoff": 35,
    },
    "susc_pp": {
        "cutoff": 35,
    },
    "hedin_ph": {
        "boson cutoff": 35,
        "fermion cutoff": 50,
    },
    "hedin_pp": {
        "boson cutoff": 35,
        "fermion cutoff": 50,
    },
    "vertex": {
        "boson cutoff": 15,
        "fermion cutoff": 35,
    },
    ...
}
```

These blocks specify the following worm observables:

- `green`: One-particle Green's functions, $G^(1)$ and $G^(1)_{imp}$

- `susc_ph`: One-time, two-particle Green's functions in the particle-hole channel $G^{(2)}_{\text{susc},ph}$
- `susc_pp`: One-time, two-particle Green's functions in the particle-particle channel $G^{(2)}_{\text{susc},pp}$
- `hedin_ph`: Two-time, two-particle Green's functions in the particle-hole channel $G^{(2)}_{\text{hedin},ph}$
- `hedin_pp`: Two-time, two-particle Green's functions in the particle-particle channel $G^{(2)}_{\text{hedin},pp}$
- `vertex`: Three-time, two-particle Green's functions in the particle-particle channel $G^{(2)}$

For a single time object, the following parameter is required:

- `cutoff`: number of Matsubara frequencies or Legendre components to measure, e.g., for Matsubara frequencies $i\omega_n$, measure all frequencies with $n <$`cutoff`

For multi-time objects, two cutoffs are required:

- `boson cutoff`: number of bosonic Matsubara frequencies to measure
- `fermion cutoff`: number of fermionic Matsubara frequencies or Legendre components to measure

Finally, in this example, we will compute the asymptotic form of the full vertex. Therefore, we add a final block

```
{
        ...
    "kernels": {
        "asymptotic cutoff": 10,
        "boson cutoff": 35,
        "fermion cutoff": 100,
    }
    ...
}
```

In the `kernels` block, we specify the following parameters:

- `asymptotic cutoff` [int]: replace measured vertex its asymptotic form if $l^4 < (\delta_{nn'}+\delta_{m0}-\delta_{m0}\delta_{nn'})|n(n-m)n'(n'-m)|$, where l iis the specified `asymptotic cutoff` (See Ref. [3].)

- `boson cutoff` [int]: number of bosonic Matsubara frequencies for which to compute the the kernel function
- `fermion cutoff` [int]: number of fermionic Matsubara frequencies for which to compute the the kernel function

Note that these input blocks can be placed in any order. Similarly, the parameters inside a block can be placed in any order. With these input parameters and a hybridization file, we are ready to run ComCTQMC and collect the results.

7.1.2 Running the example

.

As discussed in Sec. 4, this is accomplished serially by executing

```
$ $(BIN_DIR)/CTQMC params
```

or in parallel with 100 CPU's via

```
$ mpirun -n 100 $(BIN_DIR)/CTQMC params
```

where `BIN_DIR` is the location of the ComCTQMC executable. Of course, `mpirun` should be replaced by whatever mpi command is implemented on the user's computer system, e.g., `sbatch`.

Once the CTQMC portion is complete. One must execute the post-processing program `EVALSIM`. This is accomplished via the command

```
$ $(BIN_DIR)/EVALSIM params
```

7.1.3 Results

Now, let us go over the results, which were measured using 10 threads on the Intel i9 CPU (2.9 GHz), without GPU acceleration. (For a single-band model, a GPU offers essentially no speedup as the Hilbert space is too small.) Each figure is produced using a different simulation limited to only those worm spaces shown.

Figure 1 shows the results for the self-energy, the most difficult and important of the one-particle observables to measure. These results were collected in a single 15 minute simulation. The worm and partition space measurements are compared using a single, 15 minute long simulation.

As shown, the non-improved partition space measurements are comparable to the improved worm space counterparts, while the improved partition space measurement

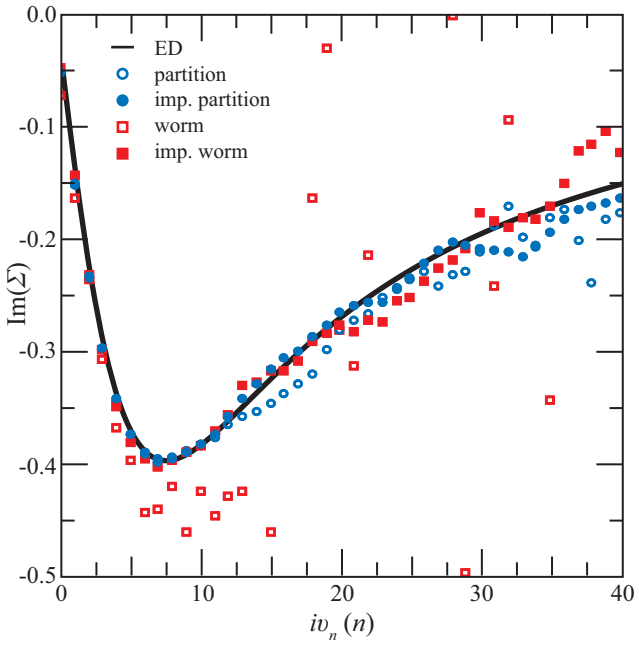

Figure 1: The local self energy in the Hubbard model at $\beta = 10$. The exact solution is compared with the measurements taken in both worm and partition spaces, with and without the use of improved estimators. The partition space measurements are more accurate, even at moderate expansion orders. (Here, $\langle k \rangle \approx 17$.) The improved estimators offer significant improvements, particularly when the results are only barely converged or for high frequencies.

is the most accurate. In general, as the temperature falls, the partition space measurements increasingly outperform the worm space measurements. As the temperature rises, the worm space measurements begin to outperform the worm space measurements. Near the atomic limit, the partition space measurements become nearly impossible while the worm measurements converge quickly. Therefore, *it is typically best practice to use the partition space measurement of the self-energy*: Not only are the partition space measurements better in the regimes wherein CTQMC is difficult, i.e., low temperatures, but each worm space sampled slows the convergence of all other observables sampled by the CTQMC solver. That said, the one-particle worm spaces can drastically improve performance if you are near the atomic limit (at high temperatures, low hybridisation strengths, or, equivalently, at low expansion order, $\langle k \rangle < 4$) where the partition space sampling becomes unreliable.

Figure 2 shows the results for the local two-point susceptibilities, measured in a 30 minute simulation. In contrast with most objects, the static susceptibilities are

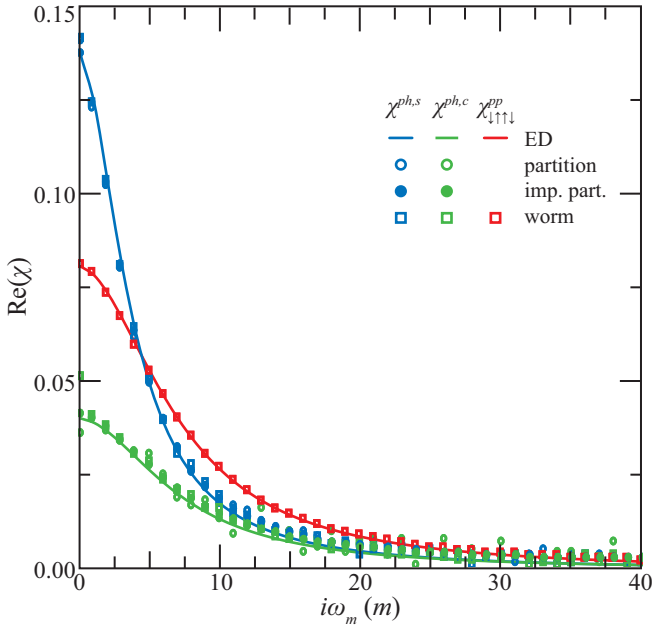

Figure 2: The local two-point susceptibilities in the Hubbard model at $\beta = 10$. The exact solution is compared with the measurements taken in both worm and partition spaces and with and without improved estimators, when possible. The static two-point susceptibilities in the particle-hole sector are difficult resolve, as are the high-frequency domains of all susceptibilities. The improved estimators help with both regions.

difficult to resolve in the particle-hole sector. Similar to all objects, the high-frequency region is also difficult to resolve. Fortunately, the high-frequency region error does not diverge, as in the self-energy or four-point vertex functions, but simply becomes noisy.

As shown, the partition-space measurements converge approximately as fast as the worm-space measurements. Note that in the Hubbard model all of the susceptibilities in the particle-hole sector can be measured in partition space. As we have discussed, this is not true in multi-channel models: When there are multiple bands, we have many $\chi_{ijkl} \neq 0$ for $i \neq j$ and $k \neq l$. However, if the hybridisation matrix is diagonal, CTQMC can only measure susceptibilities of the form χ_{iikk} while in partition space. Furthermore, ComCTQMC cannot measure any of the particle-particle sector susceptibilities in partition space, and the partition-space measurements will fail near the atomic limit. Therefore, it is best practice to use the worm space measurements for the two-particle quantities, and combine the results with the partition space measurements.

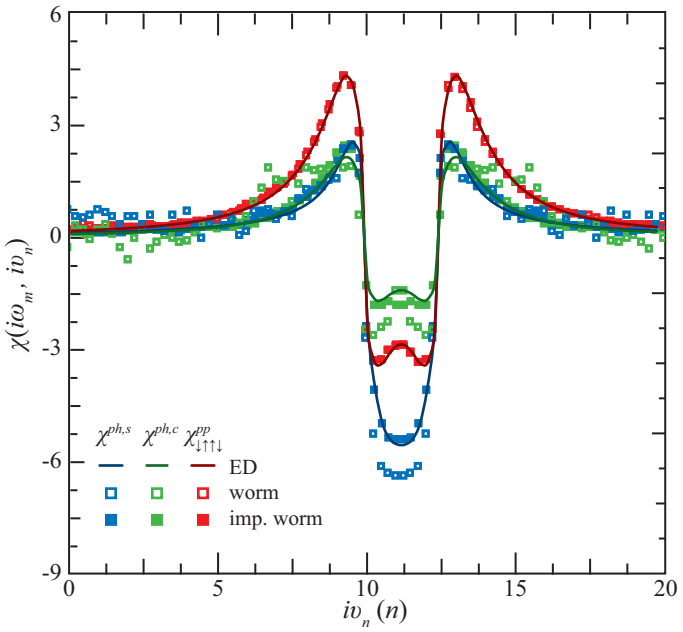

Figure 3: The local three-point susceptibilities at $i\omega_{10}$. Note that partition space measurements are not imiplemented for these objects. Improved estimators greatly increase the accuracy of the measurement, particularly at low ($0 < \nu < \omega$) and high frequencies.

Figures 3 shows the results for the two-particle, three point susceptibilities at a few bosonic frequencies. Here, partition space measurements are not implemented, but we do compare the exact results with improved and non-improved estimators. Again we see that the improved estimators drastically improve the results, particularly at high frequencies. As with the two-point susceptibilities, the low-frequency region, $0 < \nu < \omega$ can also be difficult to resolve, and one should test convergence at both high and low frequencies. The high-frequency Fermionic domain can be resolved using the Legendre basis by setting `basis = "legendre"` in the `hedin` block of the parameter file. However, as with all Legendre basis measurements, one must be careful: The error is systematic rather than stochastic, and it is therefore not as readily apparent. Furthermore, if one is computing a Hedin vertex from these three-point susceptibilities, these systematic errors can lead to a large error.

Figure4 shows the results for the full four-point vertex in the charge channel. (The full vertex is the four point susceptibility with its legs amputated, $F_{ijkl} = \chi_{ijkl}/G_iG_jG_kG_l$, and is a two-particle analogue to the self-energy.) The four point vertices require by far the most computation time for a good estimate. Here we present results using im-

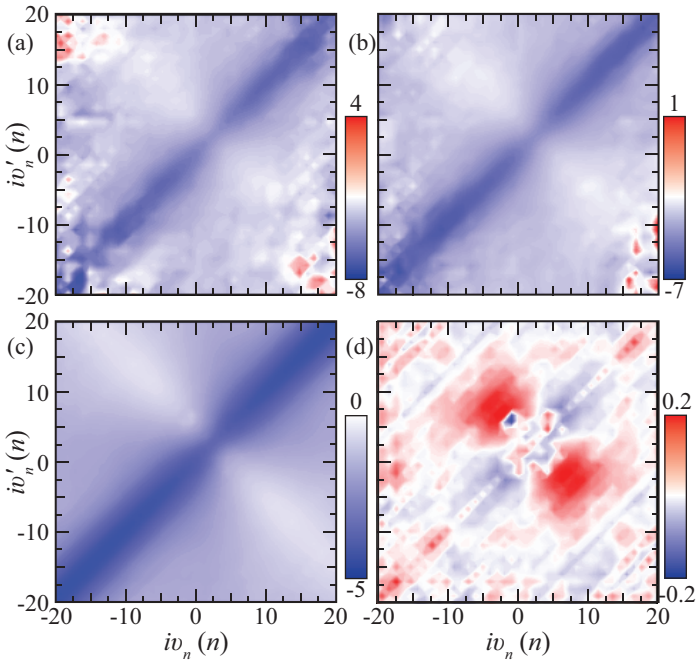

Figure 4: The real part of the local full vertex at $i\omega_5$ with the (a) Fourier and (b) mixed Legendre-Fourier basis, and (d) using the vertex asymptotics. (d) The difference between the (c) and the exact, ED result. Improved estimators are used for every observable but the local two-point susceptibilities and results are accumulated across 20,000 CPU hours. Even for this simple problem, simulated for a comparatively long time and using improved estimators, the asymptotics are crucial when resolving the vertex at an high frequencies.

proved estimators accumulated across simulations requiring nearly 20,000 CPU hours. Still, the error in the vertex at high frequencies is unacceptable, even in the Legendre basis. By using the vertex asymptotics, however, we can very nearly recover the exact, ED result at arbitrary frequencies.

Note that in order to compute the vertex asypmtotics, we must measure the two and three point susceptibilities in both particle-hole and particle-particle channels. Therefore, one must sample six configuration spaces (instead of just two) in order to compute accurate vertex functions with the correct asymptotics. In some sense, this magnifies the computational burden. However, the benefits greatly outweigh the costs, particularly if one requires the high-frequency components, as shown in Fig. 4. Still, it can be better to avoid the asymptotics, particularly if one only requires the static, $\omega = 0$, vertex.

Alternately, one can sample the full vertex within the mixed Legendre-Fourier basis. This allows ComCTQMC to evaluate the full vertex at arbitrary Fermionic frequencies, without measuring all five two-particle worm spaces in order to capture the vertex asymptotics. Despite this allowing the Markov chains to spend three times as many steps exploring the four-point observable space, the high-frequency behavior is still much worse than that captured using vertex asymptotics. Furthermore, the Legendre basis leads to structured errors. However, the user may want to experiment with the Legendre basis if only the low-frequency region is desired or if memory is constrained[2]. To use this basis, one writes the vertex block as follows:

```
{
        ...
    "vertex": {
        "basis": "legendre",
        "matsubara cutoff": 100,
        "boson cutoff": 10,
        "fermion cutoff": 50,
    }
    ...
}
```

where `matsubara cutoff` describes the number of Fourier components that the pst-processing code will produce using the `fermion cutoff` Legendre components.

7.2 δ-Plutonium

In this example, we showcase the tools which make simulating real materials more convenient and the importance of GPU acceleration. To demonstrate, we will simulate δ-Plutonium at 600 K using the self-consistent LDA+DMFT ComSuite package ComDMFT[5], within which ComCTQMC is the impurity solver. The converged hybridisation and parameter file are provided in the examples section of the ComCTQMCpackage, so that the user does not need to conduct the relatively expensive LDA+DMFT calculations. In this example, we would like to call attention to three major differences between the parameter file used for Plutonium, an f-shell system, and the file we used in the previous example.

First, we simplify the input of the two-body interaction matrix, which would be very tedious to enter by hand for a system with fourteen flavors. To do so, we specify that we are using the Slater-Condon parameterisation and give the relevant F_i parameters (U, J, etc.). That is,

[2] At low temperatures, one might need many more Fourier components than Legendre components to capture the low-frequency domain.

```
{
        ...

    "hloc": {
        "one body": [...],
        "two body": {
                "parametrisation": "slater-condon",
                "F0": 4.5,
                "F2": 6.10405,
                "F4": 4.0775,
                "F6": 3.0154,
                "approximation": "none"
    },
    ...
}
```

where `approximation` can be set to `none`, wherein all components of the interaction tensor are kept, or `ising`, wherein only the $V_{\alpha\beta\beta\alpha}$ and $V_{\alpha\beta\alpha\beta}$ components are kept.

Second, we must specify the basis used to generate the impurity problem. That is, we must provide the shell, s, p, d, or f, of the correlated orbitals and indicate whether the `product` or (spin) `coupled` basis were used. The product and coupled basis sets and their default enumeration are defined in Appendix 5. If needed, one can transform these basis sets by providing the unitary transformation matrix, as described in Appendix 4.

Here we are dealing with the f orbitals with spin-orbit coupling. In the parameter file, we specify these in the basis block as

```
{
...
"basis": {
        "orbitals": "f",
        "type": "coupled"
    },
...
}
```

Finally, we must set a flag so that ComCTQMC uses the GPU efficiently. In particular, we must describe the number of Markov chains (or simulations) an accelerated CPU should handle. Here, we have conducted scaling tests that show a CPU should handle around 25 simulations for optimal performance on Summit. Therefore, we add the following parameter to main parameter block of `params.json`:

```
{
```

```
...
"sim per device": 25,
...
}
```

In general, one should first test the scaling of their problem on their cluster to optimize this parameter, as it can drastically effect performance. That said, the GPU implementation will dynamically remove simulations from the GPU if it runs out of memory, finding the optimum number. (The number of Markov chains across all GPUs and CPUs in output in `params.info.json`; the user can figure out what the optimum number is from this result.) (This parameter is ignored if no GPUs are available.)

The properties of Plutonium (and other actinides) have been linked to the valence configuration and fluctuations within this configuration.[6]. Therefore, we direct ComCTQMC to compute the the probability a state with the specified quantum numbers is occupied. To do this, we must specify the quantum numbers in which we are interested, e.g., the total number of particles, N, and the total angular momentum in the z-direction, J_z. Additionally, we want to know the energy of the associated states. Therefore, we add the following parameters to the partition block:

```
{
...
        "partition" : {
                ...
                "quantum numbers" : {
                        "N" : { },
                        "Jz": { }
                }
                "probabilities" : ["energy","N","Jz"],
                ...
        }
...
}
```

The `energy` entry in `probabilities` directs the post-processing code to not only output the occupation probability of each state but to also output its energy. This allows us to create an energy - probability histogram.

Now, let us discuss the results.

7.2.1 Results

The LDA+DMFT simulation was run for 10 iterations on 100 nodes on Summit at ORNL. Each iteration required only 10 minutes of measurement by ComCTQMC, with

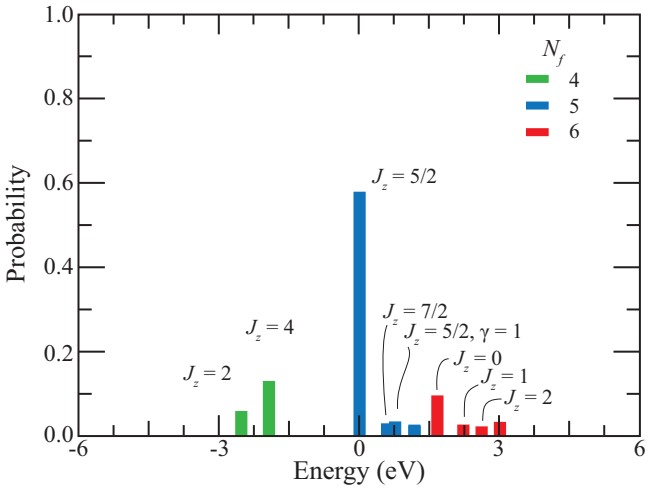

Figure 5: Valence histogram of δ-Pu at 600 K. Valence fluctuations between the $N = 5$ ground state and the $N = 6$ excited states help to explain the anomalous properties of δ-Pu.[6]

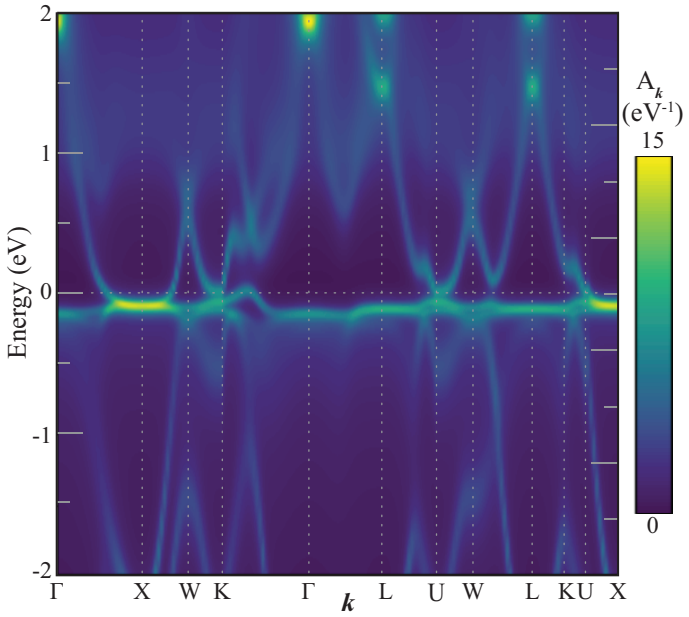

Figure 6: The momentum resolved spectral function, $A_{\boldsymbol{k}}$, of δ-Pu at 600 K along the high-symmetry lines. The spectrum features Hubbard and quasiparticle bands, with a Kondo peak just below the Fermi level.

the first iteration given another 5 minutes of thermalisation. An additional 20 minutes of measurement time was allocated to the final two iterations for the purpose of gathering more refined results for the analytical continuation.

In Fig. 5, we present the valence histogram resulting from these calculations. As has been discussed in the literature, δ-Pu exhibits significant valence fluctuations between the ground-state valence, $N = 5$ and $J_z = 5/2$, and a number of other configurations, but primarily a number of higher energy, $N = 6$ states. This differs substantially from, e.g., curium, which contains a single peak in the histogram (the ground state) with a probability near unity [6]. This valence histograms helps to explain the anomalous behavior of Plutonium, e.g., its magnetism, and the ability to quickly and easily produce such a histogram is enormously useful in the effort to understand various strongly correlated materials.

In Fig. 6, we present the self-energy and spectral function of δ-Pu. To produce the spectral function, we must first analytically continue the imaginary domain data produced by ComCTQMC. ComCTQMC does not have a built-in analytical continuation program, as a number of well-developed, open-source analytical continuation codes have already been developed and published. Here we use the maximum entropy code bundled into the extended-DMFT package (EDMFT)[7]. As shown in Fig. 6, the spectral function of δ-Pu is dominated by a Kondo peak near the Fermi level (which does not appear in pure LDA or GGA calculations).[8]

References

[1] R. Adler, G. Kotliar, Bringing electronic structure codes into the modern software ecosystem, unpublished x (2020) xx.

[2] P. Semon, C. Melnick, G. Kotliar, Gpu acceleratated ct-hyb ctqmc with worm sampling, Computational Phys. Comm. x (in preparation) xx.

[3] J. Kaufmann, P. Gunacker, K. Held, Continuous-time quantum monte carlo calculation of multiorbital vertex asymptotics, Phys. Rev. B 96 (2017) 035114. doi:10.1103/PhysRevB.96.035114.
URL https://link.aps.org/doi/10.1103/PhysRevB.96.035114

[4] A. Georges, G. Kotliar, W. Krauth, M. Rozenberg, Dynamical mean-field theory of strongly correlated fermion systems and the limit of infinite dimensions, Rev. Mod. Phys. 68 (1996) 13. doi:10.1103/RevModPhys.68.13.

[5] S. Choi, P. Semon, B. Kang, A. Kutepov, G. Kotliar, Comdmft: A massively parallel computer package for the electronic structure of correlated-

electron systems, Computer Physics Communications 244 (2019) 277 – 294. doi:https://doi.org/10.1016/j.cpc.2019.07.003.
URL http://www.sciencedirect.com/science/article/pii/S0010465519302140

[6] J. H. Shim, K. Haule, G. Kotliar, Fluctuating valence in a correlated solid and the anomalous properties of δ-plutonium, Nature 446 (7135) (2007) 513–516. doi:10.1038/nature05647.
URL https://doi.org/10.1038/nature05647

[7] K. Haule, C.-H. Yee, K. Kim, Dynamical mean-field theory within the full-potential methods: Electronic structure of ceirin$_5$, cecoin$_5$, and cerhin$_5$, Phys. Rev. B 81 (2010) 195107. doi:10.1103/PhysRevB.81.195107.
URL https://link.aps.org/doi/10.1103/PhysRevB.81.195107

[8] S. Y. Savrasov, G. Kotliar, E. Abrahams, Correlated electrons in δ-plutonium within a dynamical mean-field picture, Nature 410 (6830) (2001) 793–795. doi:10.1038/35071035.
URL https://doi.org/10.1038/35071035