ComCTQMC User Guide

Corey Melnick and Patrick Sémon

May 13, 2020

1 Introduction

ComCTQMC is a quantum impurity solver which uses the continuous time quantum Monte Carlo (CTQMC) algorithm where the action of the quantum impurty is expanded in terms of the hybridisation functions (CT-HYB). It is a stand-alone impurity solver and also embedded in dynamical mean field theory (DMFT) 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 600x 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.)

In this user guide, we will outline the quantum impurity problem solved by ComCTQMC (Sec. 2), describe how to use and install ComCTQMC (Sec. 3), list and describe the parameters which can be supplied to CTQMC (Sec. 4), and the basis available are described in (Sec. 5). Example runs are provided with the distribution in `ComCTQMC/examples/`, and described in the last section (Sec. 6)

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, $\hat{H}_{\text{bath},f}$ and $\hat{H}_{\text{bath},b}$; and the hybridisation between the the baths and the impurity, $\hat{H}_{\text{hyb},f}$ and $\hat{H}_{\text{hyb},b}$. That is,

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

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

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

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

$$\begin{aligned} S = &- \sum_{ij} \int\!\!\int_0^\beta \bar{c}_i(\tau)\mathcal{G}_{ij}^{-1}(\tau - \tau')c_i(\tau')d\tau d\tau' \\ &+ \frac{1}{2}\sum_{ijkl} \int\!\!\int_0^\beta \bar{c}_i(\tau^+)\bar{c}_j(\tau'^+)\mathcal{U}_{ijkl}(\tau - \tau')c_k(\tau')c_l(\tau)d\tau d\tau'. \end{aligned} \tag{6}$$

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

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

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 how to use ComCTQMC to solve this problem. (We refer the interested reader to Ref. [2] for a more thorough exploration of the theory behind ComCTQMC.

3 Installation and Usage

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

3.1 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 and (IBM's) Spectrum-MPI libraries.

3.2 Installation

The executables of ComCTQMC are compiled using make, and there are two executables to compile: `CTQMC` and `EVALSIM`. The Makefiles for these executables are located in `ComCTQMC/ctqmc/` and `ComCTQMC/evalsim/`, respectively. In order for the Makefiles in these directories to work, one must define which compilers and libraries to use, and one must specify where the libraries are located. The file `ComCTQMC/Makefile.in` provides fields in which to set these options, along with other compiler flags you might need to get ComCTQMC working on your computer or cluster. This file provides guidance on what these options mean and how to set them. There are also a number of examples located in `ComCTQMC/cluster_makefiles` which can be used to compile on Cori (NERSC/LBNL), Summit (OLCF/ORNL), or a Mac.

For a GPU enabled build on Summit, for example, you would do the following:

1. Set the compiler options in `Makefile.in`, e.g.,

```
#Makefile configured for Summit @ OLCF @ ORNL

#Set any basic flags for the C++ compiler, linker, and specify libraries
#required across the board
BASE_CPPFLAGS = -DNDEBUG
BASE_LDFLAGS =
BASE_LIBS = -lm

#Set flags related to the LAPACK and BLAS libraries
LAPACK_CPPFLAGS =
LAPACK_LDFLAGS = -L/$(OLCF_ESSL_ROOT)/lib64
LAPACK_LIBS = -lessl

#Set flags related to CUDA
CUDA_CPPFLAGS =
CUDA_LDFLAGS = -L/$(OLCF_CUDA_ROOT)/lib64

#Location of CUTLASS library -- included with distribution
CUTLASS_CPPFLAGS = -I../include

#Specify the compilers to use and the compiler specific flags
#Include the flag -DHAVE_MPI if the MPI enabled build is desired
CXX = g++
CXX_MPI = mpic++ -DHAVE_MPI
CXXFLAGS = -std=c++11 -fsigned-char -fno-common -O3 \
-fexceptions -Wall -Wno-sign-compare
NVCC = nvcc
NVCCFLAGS = -ccbin=$(CXX) -std=c++11 -O3 -Xcompiler  \
-fsigned-char -Xcompiler -fno-common -Xcompiler -fexceptions \
 -Xcompiler -Wall -Xcompiler -Wno-sign-compare -arch=sm_70

#Combine all of these options
CPPFLAGS = $(CUTLASS_CPPFLAGS) $(BASE_CPPFLAGS) $(LAPACK_CPPFLAGS) $(CUDA_CPPFLAGS)
LDFLAGS = $(BASE_LDFLAGS) $(LAPACK_LDFLAGS) $(CUDA_LDFLAGS)
LIBS = $(BASE_LIBS) $(LAPACK_LIBS)
```

2. Execute make in the relevant directories, e.g.,

```
cd ~/ComCTQMC/cqtmc/device/planar_complex/
```

```
make
mv CTQMC ~/ComCTQMC/bin/.

cd ~/ComCTQMC/evalsim
make
mv EVALSIM ~/ComCTQMC/bin/.
```

Note that multiple GPU builds are available for the interested user in the `ctqmc/device/` folder. The `planar_complex` directory contains the most flexible and optimal solution. (See Ref. [2] for a discussion of the other options which use MPS.)

3.3 Execute

To run ComCTQMC, one must first run the CTQMC code `CTQMC` and then run the post-processing code `EvalSim`. To do so from the base directory, one would execute

```
mpirun -np N -npernode NPN ~/ComCTQMC/bin/CTQMC params
mpirun -NP M -npernode NPN ~/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.

3.4 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 entry. 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 describe the input files required by and then output files produced by ComCTQMC.

The following input files are required by ComCTQMC:

1. `params.json`: The control file describing the simulation, the quantum impurity, the desired measurements and worms, and the system architecture.

2. `hybridisation.json`: The hybridisation functions $\Delta_{ij}(i\nu_n)$, labeled according to hybridisation block of the `params.json` file. (See Sec. 4.) Sufficient Fourier components (Matsubara frequencies) should be included so that the functions are in the asymptotic regime.

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

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 automatically continue the simulation and refine the estimators. (Produced by ComCTQMC.)

After completion, ComCTQMC produces the following output files:

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 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.

4 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 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 in a control block. Required.

beta (real, none)	Inverse temperature of the simulation
mu (real, none)	Chemical potential of the impurity.
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 steps (int, none)	Number of minutes 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.
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 historgram (boolean, true)	Accumulate a histogram of the expansion order
trunc dim (integer, ∞)	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 (matrix[real], 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 (array[real], 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 (matrix[real], I)	Transformation matrix which projects or rotates the basis such that $c_i^\dagger = \sum_j U_{ij} \tilde{c}_j^\dagger$.

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 $\Delta_{ij}(i\nu)$.
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. 6 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.
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
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)	The quantum numbers for which susceptibilities should are measured. Written as either arrays, q_n, $n = 0, 1, .., n_{\text{orbitals}}$, or as an empty dictionary, $q =$, for common quantum numbers for which ComCTQMC can auto-generate the correct arrays. Currently implemented are the particle number, N, z-component of the total spin, `Sz`, or angular momentum, `Jz`, or the total spin squared, `S2`, or total angular momentum squared, `J2`. Note that the supplied quantum numbers do not have an effect on the CTQMC algorithm or division of the Hilbert space into invariant spaces. Instead, they direct ComCTQMC to measure a particular set of susceptibilities (see the option `quantum number susceptibiility`) or associate a set of names with a set of quantum numbers for use during post-processing (see the option `probabilities`). ComCTQMC will check that the supplied quantum numbers are actually quantum numbers of the local Hamiltonian.
quantum number susceptibility (boolean, false)	The susceptibility for the quantum numbers, χ_I, are accumulated. See Sec. 6 for an example.
susceptibility cutoff (real, none)	The largest frequency to measure for the susceptibilities, $i\omega_n < 2\pi n/\beta$
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 GPUs.)
probabilities (array[string], empty)	An array listing the names of the quantum number for which the probability of occupancy should be computed during post-processing. The keyword "energy" can also be included, so that the energy of the associated state is output. Used to generate histograms of the valence occupancy. The output is sorted in ascending order across all quantum numbers. For example, `"probabilities": ["energy","N"]` directs ComCTQMC to output the energy, number of electrons, and probability of occupancy of all irreducible states of the local Hamiltonian. The output list will be ordered from lowest-to-highest energy. If two states have the same energy, they will be ordered from lowest-to-highest number of electrons.
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 and Green’s functions).
sweepB (real, 250)	Number of steps before sampling those observables which are 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)`$^4 \geq \frac{\beta^4}{\pi^4}\nu(\nu-\omega)\nu'(\nu'-\omega)$

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

$$|l, m, \sigma\rangle = Y_{l,m} \otimes \sigma, \tag{13}$$

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

$$Y_{\ell,m} := \begin{cases} i\sqrt{2}\left(Y_\ell^m - (-1)^m Y_\ell^{-m}\right) & \text{if } m < 0 \\ Y_\ell^0 & \text{if } m = 0 \\ 1\sqrt{2}\left(Y_\ell^{-m} + (-1)^m Y_\ell^m\right) & \text{if } m > 0 \end{cases} \tag{14}$$

in terms of the complex spherical harmonics Y_l^m.

The coupled basis set is defined here as

$$|j, m_j\rangle = \sum_{m\sigma} Y_l^m \otimes \sigma \langle \ell, m; \frac{1}{2}, \sigma | j, m_j \rangle, \tag{15}$$

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

$$|1, \downarrow\rangle, ..., |n, \downarrow\rangle, |1, \uparrow\rangle, ..., |n, \uparrow\rangle \tag{16}$$

$$|\ell, -\ell, \downarrow\rangle,\ |\ell, -\ell+1, \downarrow\rangle, \ldots,\ |\ell, \ell, \downarrow\rangle,\ |\ell, -\ell, \uparrow\rangle,\ |\ell, -\ell+1, \uparrow\rangle, \ldots,\ |\ell, \ell, \uparrow\rangle \tag{17}$$

$$|\ell - \frac{1}{2}, -\ell + \frac{1}{2}\rangle, \ldots,\ |\ell - \frac{1}{2}, \ell - \frac{1}{2}\rangle,\ |\ell + \frac{1}{2}, -\ell - \frac{1}{2}\rangle, \ldots,\ |\ell + \frac{1}{2}, \ell + \frac{1}{2}\rangle. \tag{18}$$

6 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 3 explains how to install and use ComCTQMC and describes all input parameters.

6.1 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

$$H = \sum_{\boldsymbol{k}\sigma} \epsilon_{\boldsymbol{k}} c^\dagger_{\boldsymbol{k}\sigma} c_{\boldsymbol{k}\sigma} + \epsilon_f \sum_\sigma f^\dagger_\sigma f_\sigma + U n_{f,\uparrow} n_{f,\downarrow} + \sum_{\boldsymbol{k}\sigma} V_{\boldsymbol{k}} (c^\dagger_{\boldsymbol{k}\sigma} f_\sigma + f^\dagger_\sigma c_{\boldsymbol{k}\sigma}), \tag{19}$$

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) [3] 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.

6.1.1 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. 3.4, 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, -0.07
            885927836, ... ],
        "real" : [ 0.01183399821, 0.006447214601, 0.005432068729, 0.00
            5127453585, ... ]
    }
}
```

    The functions in are written in Matsubara frequencies ω_n, with an array $[\text{Re}\Delta(i\omega_1), \text{Re}\Delta(i\omega_2), \dots]$ for the `real` part and an array $[\text{Im}\Delta(i\omega_1), \text{Im}\Delta(i\omega_2), \dots]$ 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. [4].)
- `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.

6.1.2 Running the example

.

As discussed in Sec. 3, 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
```

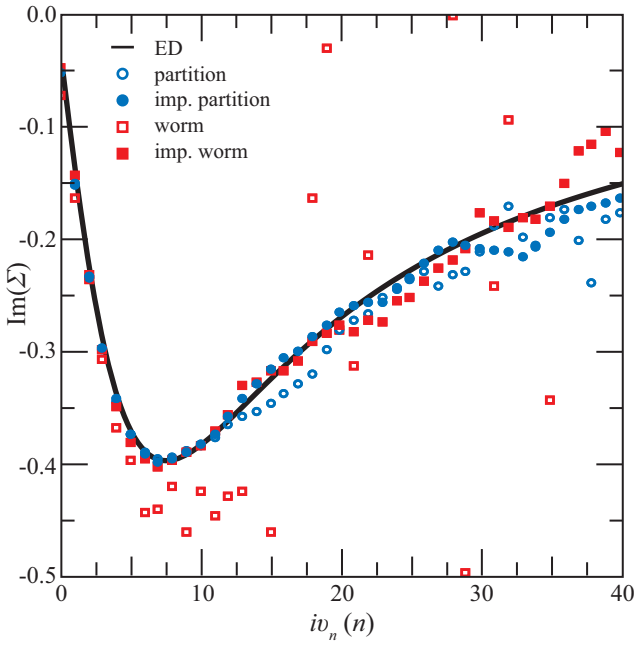

Figure 1: The local self energy in the Hubbard model at $\beta = 10$. The exact solution is compared with the measurements taken in both worm and partition spaces, with and without the use of improved estimators. The partition space measurements are more accurate, even at moderate expansion orders. (Here,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

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
```

6.1.3 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 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 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.

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

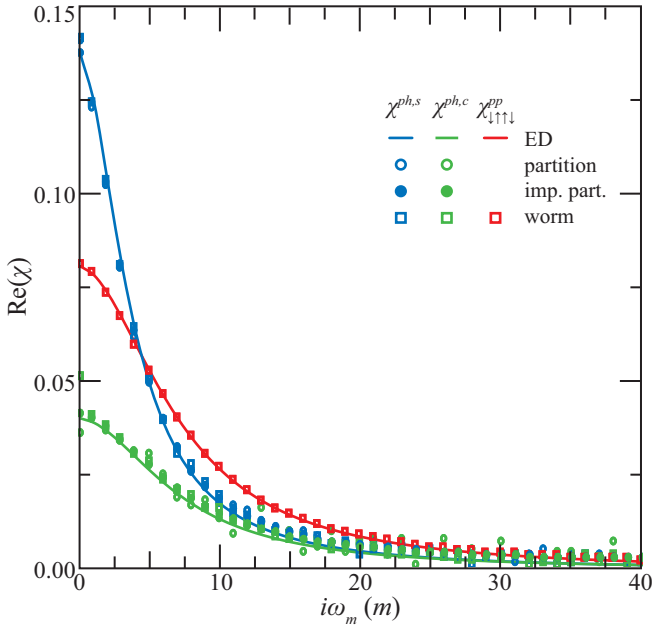

Figure 2: The local two-point susceptibilities in the Hubbard model at $\beta = 10$. The exact solution is compared with the measurements taken in both worm and partition spaces and with and without improved estimators, when possible. The static two-point susceptibilities in the particle-hole sector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

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 improved 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,

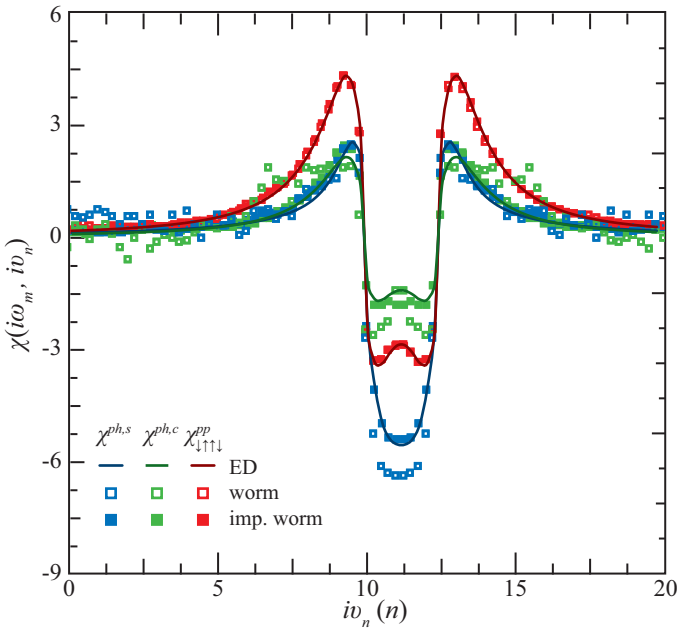

Figure 3: The local three-point susceptibilities at $i\omega_{10}$. Note that partition space measurements are not imiplemented for these objects. Improved estimators greatly increase the accuracy of the measurement, particularly at low ($0 < \nu < \omega$) and high frequencies.

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

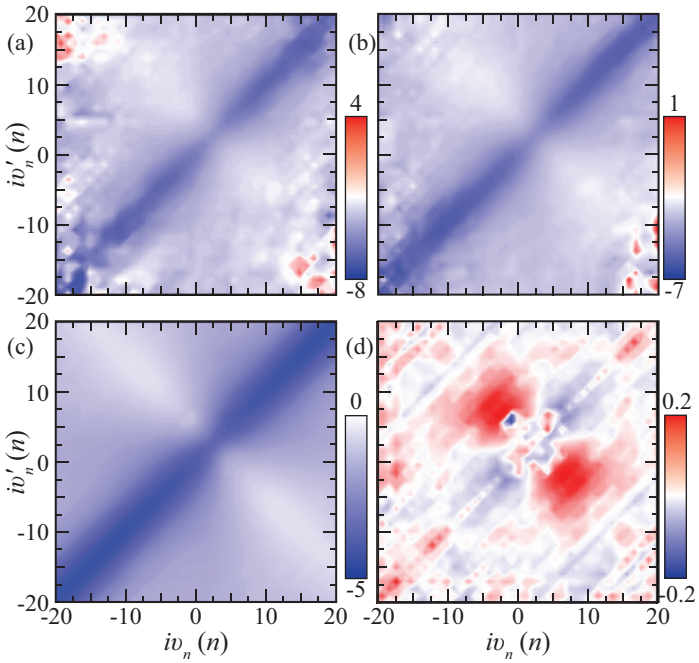

Figure 4: The real part of the local full vertex at $i\omega_5$ with the (a) Fourier and (b) mixed Legendre-Fourier basis, and (d) using the vertex asymptotics. (d) The difference between the (c) and the exact, ED result. Improved estimators are used for every observable but the local two-point susceptibilities 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.

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,
    }
```

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

```
    ...
}
```

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

6.2 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,

```
{
        ...

    "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 3.

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:

```
{
```

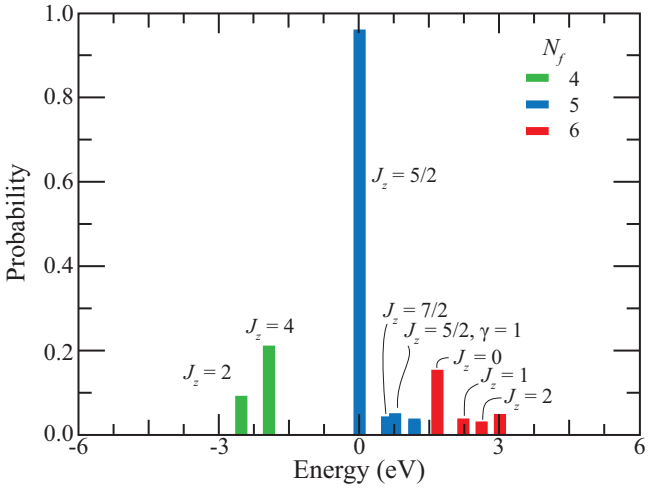

Figure 5: Valence histogram of δ-Pu at 600 K. Valence fluctuations between the $N = 5$ ground state and the $N = 6$ excited states help to explain the anomalous properties of δ-Pu.[6]

```
...
        "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.

6.2.1 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 the first iteration given another 5 minutes of thermalisation. An additional 20 min-

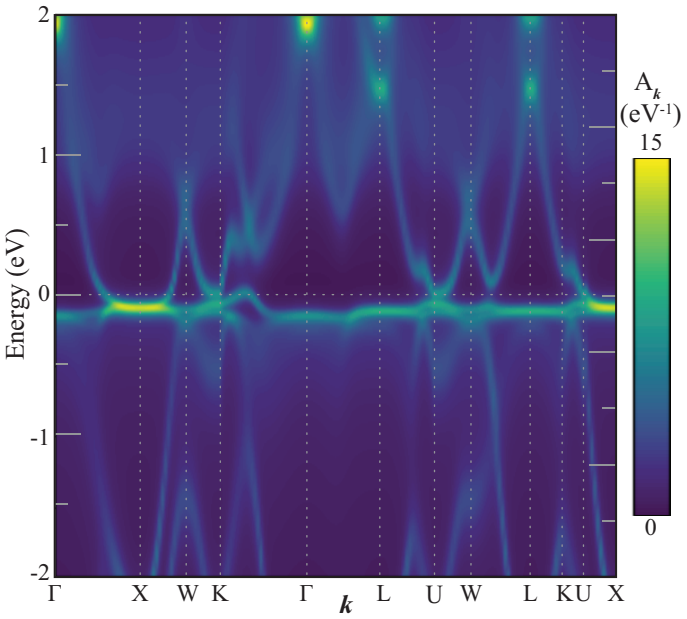

Figure 6: The momentum resolved spectral function, $A_{\boldsymbol{k}}$, of δ-Pu at 600 K along the high-symmetry lines. The spectrum features Hubbard and quasiparticle bands, with a Kondo peak just below the Fermi level.

utes 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 spec-

tral 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] 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.

[4] 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

[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