EDIpack2: interoperable Lanczos-based solver for generic quantum impurity problems

L. Crippa[a,b,c], I. Krivenko[a], S. Giuli[d], G. Bellomia[d], ...[a,d,c,d,e,f,g,h,i], A. Amaricci[i]

[a] *I. Institute of Theoretical Physics, University of Hamburg, Notkestrasse 9, 22607 Hamburg, Germany*
[b] *Würzburg-Dresden Cluster of Excellence ct.qmat, 01062 Dresden, Germany*
[c] *Institut für Theoretische Physik und Astrophysik,Universität Würzburg, 97074 Würzburg, Germany*
[d] *Scuola Internazionale Superiore di Studi Avanzati (SISSA), Via Bonomea 265, 34136 Trieste, Italy*
[e] *Politecnico di Torino, Turin, Italy*
[f] *Department of Quantum Matter Physics, University of Geneva, Quai Ernest-Ansermet 24, 1211 Geneva, Switzerland*
[g] *Department of Physics "E. Fermi" University of Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy*
[h] *LPEM, ESPCI Paris, PSL Research University, CNRS, Sorbonne Université, 75005 Paris, France*
[i] *CNR-IOM, Istituto Officina dei Materiali, Consiglio Nazionale delle Ricerche, Via Bonomea 265, 34136 Trieste, Italy*

Abstract

Keywords: Exact diagonalization, Quantum Impurity models, Strongly correlated electrons, Dynamical Mean-Field Theory

PROGRAM SUMMARY

Program Title: EDIpack2
Licensing provisions: GPLv3
Programming language: Fortran, Python
Classification: 6.5, 7.4, 20
Required dependencies: CMake (>= 3.0.0), Scifortran, MPI
Nature of problem:.
Solution method: .

Contents

1. Introduction and Motivation

SecIntro

A few words about the motivations who led us to develop this software, possible applications and advantages:

* flexibility: can address generic cases including multi-orbital, superconducting or spin-non-conserving regimes

* zero and low finite temperatures

* direct access to (well approximated) analytic dynamical functions

* direct access to impurity Fock space quantities

bla bla bla

2. Installation

SecInstall

The installation of EDIpack2 is available through CMake which ensures multi-platforms compabitility and dependencies resolution. The software builds into two distinct libraries. The main one is `libedipack.a` which, alongside the generated Fortan modules, wraps the EDIpack2 software possibly including support for inequivalent impurities. A second dynamic library, `libedipack_cbinding.so`, enables interoperability through specific bindings to the C programming language.

2.1. Structure

sSecInstallStructure

EDIpack2 is a modular library which contains three principal structures. At the core is the exact diagonalization solver: EDIpack2 . Next there is a `EDIpack2ineq` which extends application to the case of multiple inequivalent impurity problems. Finally, there we provide a Fortran-C/C++ interface, which enables development of additional API or interoperability with other, external, software.

- **EDIpack2.** This constitutes the building block of the whole software. This part implement the with the Lanczos-based solver for generic quantum impurity systems encoding different symmetries, i.e. quantum number conservations and apt to solve multi-orbital problems, also in presence of coupling to local phonons. The EDIpack2 solver has a hierarchical and modular structure: different sections of the library communicate through a shared memory layer. The top module of the library is `EDIPACK2` which, once loaded, enables access to the Fortran API in terms of suitable procedures to initialize, execute and finalize the solver or to retrieve internal quantities while making opaque to the user the internal structure of the library. A detailed presentation of the library can be found in Sec.3.

- **EDIpack2ineq.** This part of the software, leveraging on the object oriented concepts available in modern Fortran, aims to extend the EDIpack2 library to the case of multiple inequivalent and independent impurity problems. This is particularly useful while using EDIpack2 as a solver for DMFT in presence of unit cells with inequivalent atoms, for systems with somehow broken translational symmetry (e.g. heterostructures, large supercells, etc.).

- **EDIpack2 C-bindings.** EDIpack2 includes a single module implementing a Fortran-C/C++ interface of the main library procedures. The module is developed around the implicit `ISO_C_BINDING` capabilities of the most recent Fortran distributions, which enable to translate Fortran procedures directly to C. In order to overcome all the difficulties related to the internal structure of the library, we interfaced all and just the procedures and the variables exposed to the user. This module aims to foster interoperability of EDIpack2 with different third party softwares as well as to support development of additional API.

2.2. Dependencies

sSecInstallDependencies

EDIpack2 essentially depends on two external libraries.

- **SciFortran**: an open-source Fortran library to support mathematical and scientific software development.

- **MPI** (optional): a distributed memory parallel communication layer with support to modern Fortran compiler.

SciFortran provides a solid development platform enabling access to many algorithms and functions, including standard linear algebra operations and high-performance Lanczos based algorithms. This greatly reduces code clutter and development time. The use of distributed memory parallel environment, although optional, is required to access scalable parallel diagonalization algorithms which speed up calculations for large dimensional systems.

2.3.1. Source

The software can be installed from source as follows. The source can be retrieved directly from its GitHub repository, for instance using:

```
git clone https://github.com/edipack/EDIpack2.0 EDIpack2
```

Then, assuming to be in the software directoru, a conventional out-of-source building is performed using two different compilations backends.

- **GNU Make**
  This is the default CMake workflow:

```
mkdir build
cd build
cmake ..
make -j
make install
```

- **Ninja**

  An alternative workflow employs the Ninja building backend with Fortran support. Ninja is generally faster and automatically supports multi-threaded building:

```
mkdir build
cd build
cmake -GNinja ..
ninja
ninja install
```

The CMake configurations can be further tuned using the following variables:

Option	**Scope**	**Value**
-D**PREFIX**	Install directory	~/opt/EDIpack2/TAG/PLAT/BRANCH
-D**USE_MPI**	MPI support	True/False
-D**WITH_INEQ**	Inequivalent impurities support	True/False
-D**VERBOSE**	Verbose CMake output	True/False
-D**BUILD_TYPE**	Compilation flags	RELEASE/TESTING/DEBUG/AGGRESSIVE

The default target builds and install either the main library and the C-binding. However, a specific building for each library is available specifying the required target. A recap message is printed at the end of the CMake configuration step.

2.3.2. Anaconda

As an alternative we provide for both Linux and OSx systems installation through Anaconda packages into a virtual environment containing Python (> 3.10).

The Conda package installation procedure reads:

```
conda create -n edipack
conda activate edipack
conda install -c conda-forge -c edipack edipack2
```

which installs a bundle of Scifor and EDIpack2 libraries together with specific `pkg-config` configurations files which can be used to retrieve compilation and linking flags.

2.4. OS Loading

sSecInstallOSloading

In oder to avoid possible conflicts or require administrative privileges, the building step results get installed by default in a user home directory, specified by the CMake variable `PREFIX`. In doing so, however, one misses the chance of automatic loading into the operative system.

We offer different strategies to perform this action:

1. A CMake generated configuration file for environment module which allows to load and unload the library at any time. This is preferred solution for HPC systems.
2. A CMake generated bash script to be sourced (once or permanently) in any shell session to add EDIpack2 library to the default environment.
3. A CMake generated pkg-config configuration file to be added in the pkg-config path itself.

An automatically generated recap message with all instructions is generated at the end of the installation procedure.

3. Implementation

SecEDIpack

Here we present an overview of the implementation of the different parts of the EDIpack2 library.

3.1. The quantum impurity problem

sSecQIM

We consider a general quantum impurity problem defined by the following Hamiltonian:

$$\hat{H} = \hat{H}_{imp} + \hat{H}_{bath} + \hat{H}_{hyb} + \hat{H}_{ph} + \hat{H}_{e-ph}$$

which describes a multi-orbital interacting quantum impurity coupled to an electronic bath and to local, i.e. Holstein, phonons. We assuming for the moment that no particular symmetry holds. The impurity part of the Hamiltonian reads:

$$\begin{aligned}
\hat{H}_{imp} &= \hat{H}^0_{imp} + \hat{H}^{int}_{imp} \\
\hat{H}^0_{imp} &= \sum_{\alpha\beta\sigma\sigma'} h^0_{\alpha\beta\sigma\sigma'} d^+_{\alpha\sigma} d_{\beta\sigma'} \\
\hat{H}^{int}_{imp} &= U \sum_{\alpha} n_{\alpha\uparrow} n_{\alpha\downarrow} + U' \sum_{\alpha\neq\beta} n_{\alpha\uparrow} n_{\beta\downarrow} + (U'-J) \sum_{\alpha<\beta,\sigma} n_{\alpha\sigma} n_{\beta\sigma} \\
&\quad - J_X \sum_{\alpha\neq\beta} d^+_{\alpha\uparrow} d_{\alpha\downarrow} d^+_{\beta\downarrow} d_{\beta\uparrow} + J_P \sum_{\alpha\neq\beta} d^+_{\alpha\uparrow} d^+_{\alpha\downarrow} d_{\beta\downarrow} d_{\beta\uparrow}
\end{aligned} \tag{1}$$

Himp

where $d_{\alpha\sigma}$ ($d^+_{\alpha\sigma}$) are the destruction (creation) second-quantization operators for impurity electrons in the orbital $\alpha = 1, \dots, N_\alpha$, with N_α the number of orbitals, with spin $\sigma = \uparrow, \downarrow$ and whose occupation is described by the operator $n_{\alpha\sigma} = d^+_{\alpha\sigma} d_{\alpha\sigma}$. The non-interacting internal structure of the impurity is described by the $h^0_{\alpha\beta\sigma\sigma'}$ matrix. $\hat{H}^{int}$ describes the local multi-orbital interaction [1] which, for simplicity, we take as a generalized Hubbard-Kanamori form. The first three terms represent the density-density part of the interaction, where U is the local intra-orbital Coulomb repulsion, U' the inter-orbital one and J the Hund's coupling [1]. The remaining two terms are, respectively, the spin-exchange and pair-hopping which we considered with their respective independent couplings J_X and J_P. In the case $N_\alpha = 3$ a fully symmetric $SU(3)_{orbital} \times SU(2)_{spin} \times U(1)_{charge}$ form of the interaction is obtained by setting $U' = U - 2J$ and $J_X = J_P = J$ [1]. Different choices, preserving part of the combined symmetry group, can be made for other values of N_α [1].

The bath part and its coupling to the impurity has the form:

$$\begin{aligned}
\hat{H}_{bath} &= \sum_{p} \sum_{\alpha\beta\sigma\sigma'} h^p_{\alpha\beta\sigma\sigma'} a^+_{p\alpha\sigma} a_{p\beta\sigma'} \\
\hat{H}_{hyb} &= \sum_{p} \sum_{\alpha\beta\sigma\sigma'} V^p_{\alpha\beta\sigma\sigma'} d^+_{\alpha\sigma} a_{p\beta\sigma'} + H.c.
\end{aligned} \tag{2}$$

Hbath

where $p = 1, \dots, N_{bath}$ is an index running over a finite number of bath elements, $a_{p\alpha\sigma}$ ($a^+_{p\alpha\sigma}$) are the destruction (creation) operators for the bath electrons with index p, with orbital α and spin σ. The properties of each bath level are described by the matrices $h^p_{\alpha\beta\sigma\sigma'}$. As such any bath element can be composed of several electronic levels according the bath topology, which will be discussed further in the following. Each bath level couples to the impurity with an amplitude $V^p_{\alpha\beta\sigma\sigma'}$ which we allow to couple different orbital and opposite spins.

Finally, the electron-phonon part of the quantum impurity problems is described by the Hamiltonian terms:

$$\begin{aligned} \hat{H}_{ph} &= \sum_q \omega_{0q} b^+_q b_q \\ \hat{H}_{e-ph} &= \sum_q \sum_{\alpha\beta\sigma} g_{\alpha\beta} d^+_{\alpha\sigma} d_{\beta\sigma} (b_q + b^+_q) \end{aligned} \tag{3}$$

Hph

where $q = 1, \dots, N_q$ indexes the number of local phonons, b_q (b^+_q) are the destruction (creation) operators for the phonon q with frequency ω_{0q}. The matrix $g_{\alpha\beta}$ expresses is the electron-phonon coupling. Although feasible, dealing with more than one phonon mode becomes quickly computationally very demanding, thus in the rest of the this work we shall consider $N_q = 1$. In the following we consider a bath discretized into a number of bath degrees of freedom and a finite number of available phonons N_{ph}, to cut-off the unbounded dimensions of the local phonons Hilbert space.

Remark. As extensively discussed in **?** , the inclusion of local phonons with truncated dimensions ultimately amounts to a sequential application of the procedures defined for the electronic part, i.e. one per phonon mode. This includes a linear scaling of the dimensions with N_{ph} and of course largely limit the available degrees of freedom to describe electronic states. In the rest of the paper we focus specifically on the electronic part of the quantum impurity problem, referring to the phonons in specific sections when their presence introduces non-trivial modifications.

More specifically, we consider a system composed of $N_{imp} = 1$ impurities, i.e. a single impurity problem, N_{bath} bath elements and N_{ph} phonons. The size of the system is determined by the number of phonons (fixed) and that of *electronic* levels, i.e. levels with a local electronic Hilbert space $\mathcal{H}_e = \{|0\rangle, |\uparrow\rangle, |\downarrow\rangle, |\uparrow\downarrow\rangle\}$. Due to is internal structure, the single impurity contains $N_i = N_\alpha$ electronic levels. The total number of levels is then determined by the electronic levels in the bath N_b. This is a function of the bath topology and N_{bath}, i.e. the number of bath elements. In the simplest case each bath element corresponds to an independent electronic levels coupled to the impurity, thus $N_b \equiv N_{bath}$. We indicate with $N_s = N_i + N_b$ the total number of electronic levels.

The setup of the general quantum impurity problem is implemented in different parts of the EDIpack2 software. The dimensions of the system are controlled by input variables `Nspin`= N_σ, `Norb`= N_α and `Nbath`= N_{bath} in `ED_INPUT_VARS`. These are used to determine the variables `Ns`= N_s and N_b, defined in the global memory pool `ED_VARS_GLOBAL`, using the functions contained in `ED_SETUP`. The user can define the local non-interacting Hamiltonian $h^0_{\alpha\beta\sigma\sigma'}$ using the function `ed_set_hloc` in `ED_AUX_FUNX`. The matrix is then stored in the internal memory and shared throughout the code. On the other hand the setup of the bath matrices $h^p_{\alpha\beta\sigma\sigma'}$ requires a more involved procedure which will be illustrated in Sec. 3.5.

3.2. The Fock basis states

sSecBasis

The Fock space of the quantum impurity problems is defined as $\mathcal{F} = \mathcal{F}_e \otimes \mathcal{F}_{ph}$, with $\mathcal{F}_e = \bigoplus_{n=0}^{N_s} S_- \mathcal{H}_e^{\otimes n}$ the electronic Fock space, $\mathcal{F}_{ph} = \bigoplus_{n=0}^{N_q} S_+ \mathcal{H}_{ph}^{\otimes n}$ the phonon Fock space, $\mathcal{H}_{ph} = \{|0\rangle, |1\rangle, \dots, |N_{ph}\rangle\}$ is the local phonon Hilbert space and (S_-) S_+ the (anti-)symmetrization operator. The total dimension of the Fock space is $D = D_e \cdot D_{ph} = 4^{N_s} \cdot (N_{ph} + 1)$ making the exponential growth with the number of electron levels transparent. The quantum states in the space $\mathcal{F}$ are naturally represented in terms of occupation number formalism of the second quantization, i.e. the Fock basis. For a system of N_s electrons each Fock state reads $|p\rangle|\vec{n}\rangle$ with

$$|\vec{n}\rangle = |\vec{n}_\uparrow\rangle\vec{n}_\downarrow = |n_{1\uparrow}, \dots, n_{N_s\uparrow}, n_{1\downarrow}, \dots, n_{N_s\downarrow}\rangle$$

TabSector

ed_mode	*Quantum Numbers*	*Sector Dimension*
normal	$[N, S_z] \equiv [N_\uparrow, N_\downarrow]$	$\binom{N_s}{N_\uparrow}\binom{N_s}{N_\downarrow}$
superc	$S_z \equiv N_\uparrow - N_\downarrow$	$\sum_m 2^{N_s - S_z - 2m}\binom{N_s}{N_s - S_z - 2m}\binom{S_z+2m}{m}$
nonsu2	$N \equiv [N_\uparrow + N_\downarrow$	$\binom{2N_s}{N}$

Table 2: Ciao

where $p = 1, \dots, N_{ph}$ is the number of local phonons while $n_{a\sigma} = 0, 1$ signals the absence or the presence of an electron with spin σ at the level a. The electronic part of the Fock state $|\vec{n}\rangle$ is represented as a string of zeros and ones of length $2N_s$. Thus, any such state can be encoded in a computer using a sequence of $2N_s$ bits or, analogously, as a given integer $I = 0, \dots 2^{2N_s} - 1$ so that $|\vec{n}\rangle = |I\rangle$. Together with the basis states one defines destruction and creation operators, respectively $c_{a\sigma}$ and $c^+_{a\sigma}$, which acts on the Fock space as: $|\vec{n}\rangle$ as:

$$c_{a\sigma}|\vec{n}\rangle = \begin{cases} (-1)^{\#_{a\sigma}}|\dots, n_{a\sigma}-1, \dots\rangle & \text{if } n_{a\sigma}=1 \\ 0 & \text{otherwise} \end{cases}; \qquad c^+_{a\sigma}|\vec{n}\rangle = \begin{cases} (-1)^{\#_{a\sigma}}|\dots, n_{a\sigma}+1, \dots\rangle & \text{if } n_{a\sigma}=0 \\ 0 & \text{otherwise} \end{cases}$$

where $\#_{a\sigma} = \sum_{b\sigma' < a\sigma} n_{b\sigma'}$ takes care of the fermionic sign imposed by Pauli principle.

The Fock states and operators implementation can be found in ED_AUX_FUNX. There we define the bitwise action of generic fermionic creation and annihilation operators CDG and C, the binary decomposition bdecomp required to reconstruct the Fock state as bits sequence and other accessory functions.

3.3. Conserved quantum numbers

sSecQNs

In order to circumvent the exponential scaling of the dimensions of the problem scales it is necessary to take into account suitable symmetries. Indeed, considering operators $\mathcal{Q}$ such that $[H, \mathcal{Q}] = 0$ reduces of the Fock space into several symmetry sectors labelled by given quantum numbers $\vec{Q}$. In the context of quantum impurity problems two symmetries are often considered: i) conservation of the total occupation N and ii) the conservation of the total magnetization S_z. Although the total spin operator S^2 can also be conserved, the difficult implementation and the marginal gain makes it often non convenient to include this symmetry. In EDIpack2 we consider three different cases which are controlled by the input variable ed_mode=**normal**, **superc**, **nonsu2**.

The **normal** case deals with independent conservation of the total occupation N and the total magnetization S_z or, equivalently, the total number of electrons with spin up $N_\uparrow$ and down $N_\downarrow$. Optionally, we consider the case in which the symmetry applies separately for each orbital and spin, i.e. $\vec{N_\sigma} = [N^1_\sigma, \dots, N^{N_\alpha}_\sigma]$. This special case has been extensively discussed in **?** so it won't be covered in this work. The **superc** case deals with the conservation of the total magnetization only, so that total charge may not be conserved. This case includes a description of s-wave superconductivity, featuring intra- and inter-orbital components. Finally, the **nonsu2** case consider the conservation of the total charge, whereas the spin symmetry group is not fully conserved. Although there many possible realization of this scenario, this particularly applies to the presence of local spin-orbit coupling $\vec{L} \cdot \vec{S}$**?** , the emergence of in-plane spin ordering**?** or in-plane spin-triplet exciton condensation**? ? ?** . From a computational point of view the construction of a symmetry sector corresponds to the determination of a injective map $\mathcal{M} : \mathcal{S}_{\vec{Q}} \to \mathcal{F}$ relating the states $|i\rangle$ belonging to the sector $\mathcal{S}_{\vec{Q}}$ to the states $|I\rangle$ in the Fock space. Operatively, the map corresponds to an integer rank-1 array of dimension $D_{\mathcal{S}}$ which is the *dimension* of the sector. The table 3.3 summarizes the properties of the symmetry sectors.

The **normal** case requires a brief remark. Because of the independent conservation of $N_\uparrow$ and $N_\downarrow$, the local Hilbert space and the electronic Fock can be factorizes as, respectively, $\mathcal{H} = \mathcal{H}_\uparrow \otimes \mathcal{H}_\downarrow$, $\mathcal{F}_e = \mathcal{F}_{e\uparrow} \otimes \mathcal{F}_{e\downarrow}$.

Accordingly any Fock state is written as $|\vec{n}_\uparrow\rangle|\vec{n}_\downarrow\rangle$, the symmetry sector can be written as $\mathcal{S}_{\vec{Q}} = \mathcal{S}_{N_\uparrow} \otimes \mathcal{S}_{N_\downarrow}$ and, correspondingly, the sector map splits in two mutually exclusive ones $\mathcal{M} = \mathcal{M}_\uparrow \otimes \mathcal{M}_\downarrow$. Thus, each state $|i\rangle = |i_\uparrow\rangle|i_\downarrow\rangle$ of the sector is labelled by two integers $[i_\uparrow, i_\downarrow]$, $i_\sigma = 1, \dots, D_{\mathcal{S}_\sigma}$ such that $i = i_\uparrow + i_\downarrow D_{\mathcal{S}_\downarrow}$. The map $\mathcal{M}$ connects any such state to a Fock state $|I\rangle = |I_\uparrow\rangle|I_\downarrow\rangle$ labelled by two integers $[I_\uparrow, I_\downarrow]$ as $I = I_\uparrow + I_\downarrow 2^{N_s}$. For a thorough discussion about Fock basis organization in this case see **?** .

The presence of a symmetry induces a factorization of the Fock space, in turn inducing a block diagonal form to the Hamiltonian matrix. Each block, labelled by the quantum numbers $\vec{Q}$, has dimension $D_{\mathcal{S}(\vec{Q})}$. The sector Hamiltonian matrix $H_\mathcal{S}$ is represented in the basis $|i\rangle \in \mathcal{S}_{\vec{Q}}$ as a sparse matrix. In the **normal** case $H_\mathcal{S}$ takes a particularly symmetric form thanks to the product structure of the sector and its map **?** . The analysis of the spectrum is then reconducted to the inspection of the Hamiltonian in each symmetry sector. Should particular constraint holds the search can be limited to only particular sector, further reducing the computational cost.

Although the sectors have dimensions much smaller than the full Fock space, for large systems storing the Hamiltonian matrix in the memory can still be highly inefficient. In such cases, Krylov or Lanczos methods[2–5] can be implemented using a storage-free algorithm, performing the necessary linear operations on-the-fly. This solution has generally a negative impact on the execution time, however this can be well compensated by scaling in a distributed parallel framework.

The object `sector`, defined globally in `ED_VARS_GLOBAL`, contains all the informations characterizing the symmetry sector, its dimensions, its quantum numbers and an implementation of the map $\mathcal{M}$. The constructor/destructor are defined `ED_SECTORS` module with the function `build_sector`/`delete_sector` using different algorithms according to the nature of the quantum numbers $\vec{Q}$ as reported in the following code snippet.

normal

```
i=0
do Iup=0,2**Nbit-1
  nup_ = popcnt(Iup)
  if(nup_ /= Nups(1))cycle
  i = i+1
  H(iud)%map(i) = Iup
enddo
i=0
do Idw=0,2**Nbit-1
  ndw_= popcnt(Idw)
  if(ndw_ /= Ndws(1))cycle
  i = i+1
  H(iud+Ns)%map(i) = Idw
enddo
```

superc

```
i=0
do Idw=0,2**Ns-1
  ndw_= popcnt(idw)
  do Iup=0,2**Ns-1
    nup_ = popcnt(iup)
    sz_  = nup_ - ndw_
    if(sz_ /= self%Sz)cycle
    i=i+1
    self%H(1)%map(i) = &
        Iup+Idw*2**N
  enddo
enddo
```

nonsu2

```
i=0
do Idw=0,2**Ns-1
  ndw_= popcnt(Idw)
  do Iup=0,2**Ns-1
    nup_ = popcnt(Iup)
    nt_  = nup_ + ndw_
    if(nt_ /= self%Ntot)cycle
    i=i+1
    self%H(1)%map(i) = &
        Iup+Idw*2**Ns
  enddo
enddo
```

Alongside these definitions, this module additional functions to retrieve sector index or quantum number informations. Another set of key functions concern the application of arbitrary linear combinations of Fock operators to a given vector $|v\rangle \in \mathcal{S}$, i.e. $\mathcal{O}|v\rangle = \sum_i a_i C^{\dagger(p_i)}_{\alpha_i,\sigma_i}|v\rangle$ with $a_i \in \mathbb{C}$ and $p_i = 0, 1$. These important operations which depend on the sector informations are implemented in the `apply_op_C`, `apply_op_CDG`, `apply_Cops` functions within `ED_SECTORS`.

3.4. Classes

The use of suitable objects enormously simplifies the implementation of crucial mathematical concepts aking to the diagonalization of the quantum impurity problems. Here we discuss three main classes which are used in the code.

3.4.1. Sparse matrix

`CodeSparseMatrix`

A sparse matrix storage is performed using a dedicated custom class, contained in the `SPARSE_MATRIX` module. The class defines a `sparse_matrix_csr` object as a simplified hash-table. The keys corresponds to the rows of the matrix while the value is associated to a pair dynamical arrays, containing values and columns location of the non-zero elements of the sparse matrix. The `sparse_matrix_csr` object can be stored either serially, i.e. one copy per process, or be parallel distributed assigning a number of keys/values to each process. The elements are progressively stored in the dynamic arrays using `sp_insert_element` procedure,

ultimately making use of the Fortran intrinsic `move_alloc`. This ensures a faster execution compared to implicit reallocation, i.e. `vec=[vec,new_element]`. This solution enables to deal with the a priori unknown number of non-zero elements on each row, to optimize the memory footprint and to guarantee $O(1)$ access to any element of the matrix, which are crucial aspect to speed-up the execution of the MVP.

3.4.2. Sparse map

`CodeSparseMap`

As it will be evident in the following Sec. 3.9, there are cases in which the sector construction requires to store, for any sector state, disjoint information about the corresponding Fock state. Let's consider a Fock state $|I\rangle$. Its bit decomposition $|\vec{n}\rangle$ can be further split into chunks of bit corresponding, for instance to impurity and bath degrees of freedom as $|\vec{n}\rangle = |\vec{i}_\uparrow \vec{b}_\uparrow \vec{i}_\downarrow \vec{b}_\downarrow\rangle$. The module `ED_SPARSE_MAP` implements an hash-table `sparse_map`, which is included as an element in the `sector` object and which stores for every impurity configurations $\vec{i}_\sigma$ (key) the bath states $\vec{b}_\sigma$ (values) corresponding to it. The `sparse_map` $\mathcal{S}$ elements are constructed upon call in `build_sector` for any value of `ed_mode` using different algorithms. In the **normal** case, the sector object contains two `sparse_maps` $\mathcal{S}_{\sigma=\uparrow,\downarrow}$, in line with the spin dependent factorization of the Fock and sector states: $|J\rangle = |J_\uparrow\rangle \otimes |J_\downarrow\rangle \xleftarrow{\mathcal{M}} |j_\uparrow\rangle \otimes |j_\downarrow\rangle = |j\rangle$. The sparse maps are build as follows. For any spin state $|J_\sigma\rangle = |veci_\sigma \vec{b}_\sigma$ of the sector the key is determined by the integer I_σ corresponding to the impurity bit set $\vec{i}_\sigma$. The values are given by any integer B_σ corresponding to any bath bit set $\vec{b_\sigma}$ associated with $\vec{i}_\sigma$. Thus, any given combination of key-value reconstructs an integer J_σ representing the Fock states in the given spin sector according to the rule $J_s = I_\sigma + 2^{N_{imp}} B_\sigma$, where N_{imp} is the number of impurity bits.

In the **superc/nonsu2** case the construction of the unique sparse map is similar. For any Fock state $|J\rangle$ corresponding to sector state $|j\rangle$ we consider the four integers $I_\uparrow, B_\uparrow, I_\downarrow, B_\downarrow$ corresponding, respectively, to the bit sets $\vec{i}_\uparrow, \vec{b}_\uparrow, \vec{i}_\downarrow, \vec{b}_\downarrow\rangle$ decomposition the state. These integers fulfill the relation $J = I_\uparrow + B_\uparrow 2^{N_{imp}} + (I_\downarrow + B_\downarrow 2^{N_{imp}}) 2^{N_s}$ To obtain a contiguous memory pattern we define as a key of the sparse map the combination $I_\uparrow + I_\downarrow 2^{N_{imp}}$ and as corresponding values the integer $B_\uparrow + B_\downarrow 2^{N_b}$. Where needed the correct bit sets and Fock state can be reconstructed with simple algebra at no additional cost.

normal

```
i=0
do Iup=0,2**Nbit-1
  nup_ = popcnt(Iup)
  if(nup_ /= Nups(1))cycle
  i=i+1
  !...
  iImp  = ibits(iup,0,Norb)
  iBath = ibits(iup,Norb,Norb*Nbath)
  call sp_insert_state(self%H(1)%sp,iImp,iBath,i)
enddo
i=0
do Idw=0,2**Nbit-1
  ndw_= popcnt(Idw)
  if(ndw_ /= Ndws(1))cycle
  i=i+1
  !...
  iIMP  = ibits(idw,0,Norb)
  iBATH = ibits(idw,Norb,Norb*Nbath)
  call sp_insert_state(self%H(2)%sp,iImp,iBath,dim)
enddo
```

superc/nonsu2

```
i=0
do Idw=0,2**Ns-1
  ndw_= popcnt(idw)
  do Iup=0,2**Ns-1
    nup_ = popcnt(iup)
    !superc
    sz_  = nup_ - ndw_
    if(sz_ /= self%Sz)cycle
    !nonsu2
    !nt_  = nup_ + ndw_
    !if(nt_ /= self%Ntot)cycle
    i=i+1
    !...
    iImpUp  = ibits(iup,0,Norb)
    iImpDw  = ibits(idw,0,Norb)
    iBathUp = ibits(iup,Norb,Norb*Nbath)
    iBathDw = ibits(idw,Norb,Norb*Nbath)
    iImp    = iImpUp  + iImpDw*(2**Norb)
    iBath   = iBathUp + iBathDw*(2**(Norb*Nbath))
    call sp_insert_state(self%H(1)%sp,iImp,iBath,dim)
  enddo
enddo
```

3.4.3. Eigenspace

`CodeEigenspace`

The module `ED_EIGENSPACE` implements a list storing for the eigenvalues and eigenvectors of the quantum impurity Hamiltonian. This class defines the object `sparse_espace`, an ordered linked list storing the eigenvalue (the sorting key), the eigenvector and the corresponding QNs. To save memory the eigenvectors are automatically distributed to all processors in shares of the right size according to the nature of the quantum numbers $\vec{Q}$.

For zero temperature calculations only the groundstates (with degeneracies) are stored in the list. For a finite temperature the excited states need to be stored too. In order to avoid unbounded growth of the list we adopt an truncation mechanism. In the first call we collect a number `lanc_nstates_sector` of states from each sector, up to a given maximum number `lanc_nstates_total`, both set on input. The list is truncated by keeping the states which fulfil the condition $e^{-\beta(E_i-E_0)} <$ `cutoff`, where E_i is the energy of the i^{th} state in the list, E_0 is the groundstate energy, $\beta = 1/T$ is the inverse temperature ($k_B = 1$) and `cutoff` is an input parameter fixing an a priori energy threshold. Annealing is achieved by successive diagonalization of the problem. The numbers of states required to any sector $\mathcal{S}$ contributing to the list is increased by `lanc_nstates_step` or it is reduced otherwise. After few calls (of the order of ten) the distribution among the sectors of the numbers of states reaches a steady state. The corresponding annealed list contains all and just the states contributing to the spectrum up to the required energy threshold. A histogram of the number of states for each sector is produced after each diagonalization to check the evolution of their distribution.

3.4.4. GFmatrix

One of the main goal of the code is to evaluate dynamical correlations functions (DCF) $\langle \mathcal{T}[A(t)A^+]\rangle$. As we shall see in the following, using Krylov method it is possible to express the dynamical correlations in terms of a suitably truncated Kallen-Lehmann spectral sum of the form $\frac{1}{Z}\sum_n e^{-\beta E_n}\sum_{m=1}^{N}\frac{|w_{mn}|^2}{z-dE_{mn}}$ where w_{mn} is a weight set by the projection of the m^{th} eigenstate onto the n^{th} component of the Krylov basis (which reduce the sector Hamiltonian into a partial tri-diagonal form) while $dE_{mn} = E_m - E_n$ is an excitation energy. The `ED_GFMATRIX` module contains a class which efficiently store all the *weights* w_{mn} and *poles* dE_{mn} contributing to a specific correlation function. Specifically, the `gfmatrix` object implements a dynamical multi-layer data structure storing any DCF as the set of all the weights and poles, from any contributing eigenstate and for any combination of operators appearing in its definition. The use of this class enables the istantantaneous evaluation of a given DCF $\mathcal{G}(z)$ for any $z \in \mathbb{C}$ the complex frequency plane using a suitable compressed form.

3.5. Bath parametrization

`sSecBath`

The bath parametrization is a key feature in determining the properties of the quantum impurity problem. Following the structure of Eq. 2 the bath can be parametrized by two terms: the Hamiltonian matrices h^p and the amplitudes V^p, for $p = 1, \ldots, N_{bath}$. Internally, the bath is represented by a dedicated object `effective_bath` defined in `ED_VARS_GLOBAL`. On the user side all parameters are packed into a rank-1 array of doubles handled using reverse communication strategy. This ensures that a local array is always conserved by the user, while it prevents direct access to the internal copy.

The bath topology, i.e. the links between the N_b electronic levels assigned to the bath, is determined by the input variable `bath_type` among 4 different choices: **normal**, **hybrid**, **replica** and **general** (see Fig.**??**).

`bath_type`=**normal**. The bath is formed out of N_{bath} electronic levels coupled to each of the impurity orbital. The total count of bath levels is $N_b = N_\alpha N_{bath}$. The bath Hamiltonan matrices include a parametrization diagonal in the orbital and spin space: $h^p_{\alpha\beta\sigma\sigma'} = \varepsilon_{\alpha\sigma}\delta_{\alpha\beta}\delta_{\sigma\sigma'}$. If `ed_mode`=**superc** one needs to account for anomalous amplitudes connecting bath levels with opposite spins. So we consider an additional set of parameters, diagonal in orbital space but off-diagonal in the spins: $h^p_{\alpha\beta\sigma\sigma'} = \Delta^p_\alpha\delta_{\alpha\beta}\delta_{\sigma\bar{\sigma}}$. This choice corresponds to consider, for every orbital component, bath matrices with the following structure in the Nambu space: $\hat{h}^p_{\alpha\beta} = \varepsilon_\alpha\delta_{\alpha\beta}\tau_0 + \Delta_\alpha\delta_{\alpha\beta}\tau_x$. The hybridization amplitudes between the impurity and the bath levels include, for any value of `ed_mode` a set of parameters diagonal in both spin and orbital space: $V^p_{\alpha\beta\sigma\sigma'} = V^p_{\alpha\sigma}\delta_{\alpha\beta}\delta_{\sigma\sigma'}$. If `ed_mode`=**nonsu2** the an additional set including terms describing spin-flip processes (as total magnetization is not conserved) should be included $V^p_{\alpha\beta\sigma\sigma'} = W^p_\alpha\delta_{\alpha\beta}\delta_{\sigma\bar{\sigma}}$.

`bath_type`=**hybrid**. The bath is formed out of N_{bath} coupled to all the impurity levels, correspondingly $N_b \equiv N_{bath}$ is the total count of electronic bath levels. The main parametrization of the bath Hamiltonian matrices is still diagonal in both orbital and spin space, i.e. $h^p_{\alpha\beta\sigma\sigma'} = \varepsilon_{\alpha\sigma}\delta_{\alpha\beta}\delta_{\sigma\sigma'}$. As for the previous

case, the **superc** mode requires the inclusion of a further set of parameters taking care of the anomalous components: $h^p_{\alpha\beta\sigma\sigma'} = \Delta^p_\alpha \delta_{\alpha\beta}\delta_{\sigma\bar{\sigma}}$. The key difference with respect to the **normal** case lies in the off-diagonal nature of the hybridization amplitudes: $V^p_{\alpha\beta\sigma\sigma'} = V^p_{\alpha\beta\sigma}\delta_{\sigma\sigma'}$. If total magnetization is not conserved, i.e. for `ed_mode`=**nonsu2**, an additional set of parameters should be included to describe spin-flip processes: $V^p_{\alpha\beta\sigma\sigma'} = W^p_{\alpha\beta}\delta_{\sigma\bar{\sigma}}$. These sets of hybridization parameters allow to capture the effects of locally hybridized impurity orbitals. This comes at the cost of a slightly harder optimization process (see Sec. 3.11) specially for systems with reduced number of available bath levels with respect to the impurity.

`bath_type`=**replica/general**. A more flexible parametrization of the bath is represented by this topology. The original idea of this approach is to give to each bath element a structure which *replicates* that of the impurity while keeping a diagonal coupling between bath elements and impurity. This offloads the difficulties related to the representation and optimization of strucured quantum impurities to the bath Hamiltonan rather than the hybridizations. Taking a sligthly more general point of view we consider a user defined matrix basis $\vec{\Gamma} = \{\Gamma^\nu_{\alpha\beta\sigma\sigma'}\}_{\nu=1,\dots,N_{sym}}$ in the (Nambu-)spin-orbital space and parametrize any bath Hamiltonian as:

$$h^p = \sum_{\nu=1}^{N_{sym}} \lambda^p_\nu \Gamma^\nu \equiv \vec{\lambda}^p \cdot \vec{\Gamma}$$

where $\vec{\lambda}^p \in \mathbb{R}^{N_{sym}}$ is a vector of variational parameters. The choice of the matrix basis can be inspired either by the internal structure of the quantum impurity, i.e. $\hat{h}^0$, or be determined case-by-case by the properties of the problem. The total number of electronic levels used to describe the bath is in this case: $N_b = N_{sym}N_{bath}$. For the **replica** topology the coupling between the impurity and each bath elements is diagonal in the spin, orbital and internal bath structure: $V^p_{\alpha\beta\sigma\sigma'} = V^p$. The **general** setup introduces a generalization introducing a dependence on the internal spin and orbital indices: $V^p_{\alpha\beta\sigma\sigma'} = V^p_{\alpha\sigma}\delta_{\alpha\beta}\delta_{\sigma\sigma'}$.

All the procedures concerning the bath, either on the user side or the internal `effective_bath`, are grouped into a set of modules wrapped by `ED_BATH`. We divided the set of modules in three categories, according to their scope.

Bath Auxiliary Tools. This part contains some functions directed to the user as, for istance, the implementation of suitable conventional symmetry operations acting on the user bath array like, e.g., orbital symmetry, particle-hole symmetry, etc. found in `ED_BATH_USER` However, the key functions of this group concerns the determination of the total dimension of the user bath array in `ED_BATH_DIM`. Starting from the values of some input variables the function `ed_get_bath_dimension` returns the dimension B to which the user should allocate the bath array to contains all and just the bath parameters. Any following call to functions hosting the bath as an input will check that the user supplied bath array has the correct dimensions compared to the value of B using `check_bath_dimension`.

Bath Replica (General). The module `ED_BATH_REPLICA` implements the class `Hreplica` for the the **replica** (**general**) bath parametrization. Beside the conventional operations of object construction, destructin, reading and saving the module includes the matrix basis $\{\Gamma^\nu\}_{\nu=1,\dots,N_{sym}}$ and variatinal parameters $\vec{\lambda}^p$ setup, i.e. `set_Hreplica` (`set_Hgeneral`) as well as a dedicated function to build the bath Hamiltonian $h^p = \vec{\lambda}^p \cdot \Gamma$ in `build_Hreplica` (`build_Hgeneral`).

Effective Bath. The module `ED_BATH_DMFT` implements a class for the object `effective_bath` (already defined in `ED_VARS_GLOBAL`). This data structure contains and handles all the actual parameters of the bath, for any choice of `ed_mode` and `bath_type`, referencing the class `Hreplica` only for the **replica/general** bath topology. A global instance of the class `dmft_bath` containing the current bath parameters is shared throughout the code. The class constructor is supplemented with a specific procedure initializing the effective bath `init_dmft_bath` which guess the bath parameters or read them from a file specified by the input variable `Hfile`. The class also contains procedures to transform the instance `dmft_bath` to/from the user bath array: `get/set_dmft_bath`.

3.6. Lanczos based Diagonalization

sSecHam

In presence of symmetries (see Sec. 3.3) the matrix representing the Hamiltonian operator $\hat{H}$ takes a block diagonal form. Each block contains the Hamiltonian matrix $H_{\mathcal{S}_{\vec{Q}}}$ for a given symmetry sector with quantum number $\vec{Q}$. The analysis of the energy spectrum of the quantum impurity problems thus reduces to the sequential solution of the sector secular problem

$$H_{\mathcal{S}_{\vec{Q}}}|v\rangle = E_{\mathcal{S}_{\vec{Q}}}|v\rangle.$$

Although the dimension of any such sector $\mathcal{S}$ are much smaller than the total Fock space dimension $D_{\mathcal{S}} \ll D_{\mathcal{F}}$ still the complete solution of the related eigenvalue problem represents an unsurmountable problem already with $N_s \simeq 8$. In order to outwit this issue a number of algorithms have been developed leveraging on the sparse nature of $H_{\mathcal{S}}$, e.g. Krylov or similar sub-space methods[? ? ?]. In EDIpack2 the alghoritm of choice is P-ARPACK, which represents the state-of-the-art Lanczos-based eigensolver with distributed memory support[?]. This method includes re-orthogonalization within a block Lanczos algorithm which ensures convergence of the requred eigenpairs. Using the input variable `lanc_method` it is however possible to select a simple, but not as efficient, parallel Lanczos algorithm. The core of Krylov-based diagonalization algorithms is the Matrix-Vector Product (MVP), i.e. the application of the Hamiltonian to a given input vector:

$$H_{\mathcal{S}}|v\rangle \to |w\rangle \qquad |v\rangle, |w\rangle \in \mathcal{S}_{\vec{Q}}$$

This operations easily takes over more than 80% of the computation time, so its optimization is a crucial aspect in any diagonalization algorithms. In [?] we covered in full details some distrubuted memory massively parallel algorithms to address this specific issue. Here, using the same naming convention introduced in[?], we briefly outline the main strategy for the implementation of the MVP according to the value of `ed_mode`.

normal. This case relies on the tensor structure of the Fock space and the symmetry sectors. In any given sector the electronic part of the Hamiltonian matrix reads:

HssNormal

$$H_{\mathcal{S}} = H_d + H_\uparrow \otimes \mathbb{I}_\downarrow + \mathbb{I}_\uparrow \otimes H_\downarrow + H_{nd}\,. \tag{4}$$

where H_d is a diagonal term containing the local part of the Hamiltonian, there including the density-density terms of the interaction, H_σ describes the hopping processes of the electrons with spin $\sigma = \uparrow, \downarrow$. The term H_{nd} contains all the remaining non-diagonal elements which do not fit in the previous two components, e.g. the spin-exchange and pair-hopping interaction terms. Correspondingly, the any sector state $|v\rangle$ can be expressed in a matrix basis $\hat{v}$ with rows (columns) running over spin $\uparrow$ ($\downarrow$) configurations. Leveraging on this setup it is possible the MVP using the `MPI_All2AllV` which aims at optimizing locality in the memory locality. Additional terms contained in H_{nd} are treated using the `MPI_AllGatherV` algorithm which includes a slight communication overload ultimately reducing the parallel efficiency.

superc/nonsu2. In this two cases the symmetry enforce a specific relation between $\uparrow$ and $\downarrow$ configurations ultimately suppressing the benefits arising from Fock space factorization. The electronic part of the Hamiltonian matrix is usually organized in this structure:

$$H_{\mathcal{S}} = H_{imp} + H_{int} + H_{bath} + H_{imp-bath}$$

where, with almost obvious meaning of the labels, H_{imp} encodes the non-interacting part of the impurity Hamiltonian determined by $h^0_{\alpha\beta\sigma\sigma'}$, H_{int} the interaction term, H_{bath} the effective bath terms related to $h^p_{\alpha\beta\sigma\sigma'}$ and finally $H_{imp-bath}$ describes the coupling between impurity and bath proportional to $V^p_{\alpha\beta\sigma\sigma'}$. In an MPI setup the first three terms contains elements local in the memory of each node. These are stored separately from the non-local ones in any `sparse_matrix` representing $H_{\mathcal{S}}$. The MVP is carried out using the `MPI_AllGatherV` algorithm which, as discussed in [?], requires the reconstruction of the distributed input array any node producing a overflow in the MPI communication.

The diagonalization procedure is divided in two main steps: a global setup and the actual diagonalization. The setup step takes care of allocating the required memory, setup the MPI environment and allocate the correct Matrix-Vector Product (MVP) procedure according to the symmetries of the problem. This part is implemented in a set of independent modules `ED_HAMILTONIAN_?`, where `ed_mode=?`. Specifically, any such module includes two functions. The first, `build_Hv_sector_?`, builds the required sector and allocate the MVP function, i.e. point the abstract function `sphtimesv_p` to the correct procedure determined by the value of `ed_sparse_H=True/False`. If `True` the Hamiltonian terms in $H_{\mathcal{S}}$ get evaluated and stored in a `sparse_matrix` instance, possibly using automatic parallel storage, and then used to perform the MVP. This part is implemented in the modules `ED_HAMILTONIAN_?_STORED_HxV` If `ed_sparse_H` is `False` the MVP is operated on-the-fly, i.e. each element of $H_{\mathcal{S}}$ is direcrtly applied to the given input vector $|v\rangle$ to determine the outgoing vector $H_{\mathcal{S}}|v\rangle$, either in serial or parallel mode as implemented in `ED_HAMILTONIAN_?_DIRECT_HxV`.

The function `vecDim_Hv_sector_?` returns the dimension of the vector to be used in the MVP. In serial execution this number is just $D_{\mathcal{S}}$ the dimension of the sector, whereas in a parallel mode the returned value is the dimension d_i of the vector chunk per each node such that $\sum_{i=1}^{N_{nodes}} d_i = D_{\mathcal{S}}$. Yet, the specific value of d_i depends on the MPI algorithm used for the MVP function, which in turn is determined by the symmetry of the problem, i.e. `ed_mode`.

The sector-by-sector diagonalization for each value of `ed_mode` is implemented in the modules `ED_DIAG_?` through the main functions `diagonalize_impurity_?`. The diagonalization proceeds in three main steps: i) setup the diagonalization by selecting all or a sub-set of the sectors to analyze; ii) sequential diagonalization cycle over all the suitable symmetry sectors; iii) analysis of the resulting `state_list` containins all the eigensolutions to be conserved.

3.7. Dynamical correlation functions

`sSecGF`

The determination of the low energy part of the Hamiltonian spectrum enables the evaluation of dynamical correlation functions using Krylov sub-space algorithm. This is a key part of the library when using EDIpack2 as an impurity solver within DMFT framework.

Before entering into the implementation details we briefly outline the generic algorithm. Let us consider the dynamical correlation function:

`eqGaa`

$$C_{\mathcal{A}} = \langle \mathcal{T}_{\pm}[\mathcal{A}(t)\mathcal{A}^{+}] \rangle \tag{5}$$

where $\mathcal{A}(t) = e^{iHt}\mathcal{A}e^{-iHt}$, $\mathcal{T}_{\pm}$ is the time-ordering operator for fermions or bosons and $\langle \mathcal{A} \rangle = \frac{1}{Z}\mathrm{Tr}\left[e^{-\beta H}\mathcal{A}\right]$, $Z = \sum_n e^{-\beta E_n}$, is the thermodynamic average. Using spectral decomposition, the expression Eq. (5) is reduced into Kallen-Lehman form:

`KLgf`

$$\begin{aligned} C_{\mathcal{A}}(z) &= \langle \mathcal{A}\frac{1}{z-H}\mathcal{A}^{+} \rangle \mp \langle \mathcal{A}^{+}\frac{1}{z+H}\mathcal{A} \rangle \\ &= \frac{1}{Z}\sum_n e^{-\beta E_n}\sum_m \frac{|\langle\psi_m|\mathcal{A}^{+}|\psi_n\rangle|^2}{z-(E_m-E_n)} \mp \frac{|\langle\psi_m|\mathcal{A}|\psi_n\rangle|^2}{z+(E_m-E_n)} \end{aligned} \tag{6}$$

where $z \in \mathbb{C}$, $|\psi_n\rangle$, E_n are the eigenpairs of the Hamiltonian H and E_0 is the groundstate energy. While appealing, the previous expression requires the knowledge of the entire Hamiltonian spectrum. A key simplification is obtained looking at the first line in Eq. (6), showing that the $C_{\mathcal{A}}$ essentially corresponds to a given element of the inverse operator $(z-H)^{-1}$, i.e. the resolvant. The Krylov sub-space method provides a controlled approximation for this element, leveraging over the reduction of the (sector) Hamiltonian to a partial tri-diagonal form and returning an arbitrary number of amplitudes to (unknown) excited states. To illustrate the algorithm let's consider the normalized initial state:

$$|\phi_n\rangle = \mathcal{A}^{+}|\psi_n\rangle/\mathcal{N}_n$$

where $|\psi_n\rangle \in \mathcal{S}$ and $\mathcal{N}_n = \sqrt{\langle\psi_n|\mathcal{A}\mathcal{A}^{+}|\psi_n\rangle}$. We can construct the Krylov basis

$$\mathcal{K}_N(|\phi_n\rangle) = \{|\phi_n\rangle, H|\phi_n\rangle, \dots, H^N|\phi_n\rangle\} \equiv \{|v_0^n\rangle, |v_1^n\rangle, \dots, |v_N^n\rangle\}$$

with $1 \ll N \ll \mathcal{D}_{\mathcal{S}}$ by iteratively applying the (sector) Hamiltonian through MVP functions. Any given eigenstate $|\psi_n\rangle$ has components along the Krylov basis $\mathcal{K}_N(|\phi_n\rangle)$ which read: $|\psi_n\rangle = \sum_i \langle v_i^n|\psi_n\rangle |v_i^n\rangle = \sum_i a_i^n |v_i^n\rangle$. We can evaluate the operator components $= \langle\psi_m|\mathcal{A}^+|\psi_n\rangle \equiv \langle\psi_m|\phi_n\rangle = \mathcal{N}_n a_m^{n*}$. Inserting this expression into Eq. (6) we obtain the following approximation to $C_{\mathcal{A}}$:

$$
\begin{aligned}
C_{\mathcal{A}}(z) &\simeq \frac{1}{Z}\sum_n e^{-\beta E_n}\sum_{m=1}^{N}\frac{\langle\psi_n|\mathcal{A}\mathcal{A}^+|\psi_n\rangle|a_m^n|^2}{z-(E_m-E_n)} \mp \frac{\langle\psi_n|\mathcal{A}^+\mathcal{A}|\psi_n\rangle|a_m^n|^2}{z+(E_m-E_n)} \\
&= \frac{1}{Z}\sum_n\sum_{m=1}^{N}\sum_{\nu=\pm}\frac{w^\nu_{mn}[\mathcal{A}]}{z-dE^\nu_{mn}[\mathcal{A}]} = \frac{1}{Z}\sum_n\sum_{m=1}^{N}\sum_{\nu=\pm} g_{\mathcal{A}}(\nu, w^\nu_{mn}, dE^\nu_{mn})
\end{aligned}
\qquad \texttt{eqGKrylov} \quad (7)
$$

where we introduced the notation $g(\nu, w(\mathcal{A})^\nu_{mn}, dE(\mathcal{A})^\nu_{mn})$ for the `gfmatrix` object containing the weights $w(\mathcal{A})^\nu_{mn}$ and poles $dE(\mathcal{A})^\nu_{mn}$ for the operator $\mathcal{A}$, for every initial state $|\psi_n\rangle$ contributing to the low energy spectrum, for every order m of the Krylov sub-space algorithm and for channel ν.

A stringent limitation of this procedure is to get applied to diagonal dynamical correlation functions of the form Eq. (5). However, as we will show in the following, many practical application requires to evaluate off-diagonal functions of the form: $C_{\mathcal{AB}}(z) = \langle T_\pm[\mathcal{A}(t)\mathcal{B}^+]\rangle$. A direct extensin of the method is obtained considering suitable auxiliary operators prepared with independent linear combination of $\mathcal{A}$ and $\mathcal{B}$. A notable example is to consider the two auxiiary operators $\mathcal{O} = \mathcal{A} + \mathcal{B}$ and $\mathcal{P} = \mathcal{A} - i\mathcal{B}$. Using simple algebra it is then straightforward to obtain the desired function $C_{\mathcal{AB}}$ from the evaluation of $C_{\mathcal{O}}$ and $C_{\mathcal{P}}$:

$$
C_{\mathcal{AB}} = \frac{1}{2}\left[C_{\mathcal{O}} + C_{\mathcal{P}} - (1-i)C_{\mathcal{A}} - (1-i)C_{\mathcal{B}}\right]
$$

In EDIpack2 the impurity Green's functions $G_{\alpha\beta\sigma\sigma'} = \langle\mathcal{T}_\pm[c_{\alpha\sigma}(t)c^+_{\beta\sigma'}]\rangle$, is contained in `ED_GREENS_NORMAL`. This wraps more specific methods according to the symmetry of the problem expressed by `ed_mode`. In the **normal** mode also spin, charge, pair and excitonic susceptibilities functions. Computationally, the construction of the Krylov basis $\mathcal{K}_N(\mathcal{O}|\psi_n\rangle)$ for any eigenstate of the low energy spectrum is the most time consuming step. As for the diagonalization part, here a crucial speed-up is obtained by the parallel execution of the MVP lying at the earth of the Hamiltonian tri-diaagonalization process. The input variable `lanc_gfniter` controls the largest order of the Krylov basis, i.e. the maximum number of excitations in Eq. (7). Operationally, each symmetry mode `ed_mode=?` requires a different construction of the Green's functions implemented in `ED_GF_?`

normal. In this case the all the orbital dependent and spin-diagonal Green's functions $G_{\alpha\beta\sigma\sigma}$ needs to be evaluated. The orbital diagonal case $\alpha = \beta$ proceeds as discussed above considering $\mathcal{A} = c_{\alpha\sigma}$. This step makes use of the `apply_op_C/CDG` functions in `ED_AUX_FUNX` on any eigenstate $|\psi_n\rangle$ in the `state_list`, i.e. a global instance of `sparse_espace` containing the low energy spectrum of the problem. The sector Hamiltonian matrix in this mode is assumed to be real symmetric. This brings a simplification in the evaluation of the off-diagonal components $G_{\alpha\beta\sigma\sigma} = G_{\beta\alpha\sigma\sigma}$. These components are thus evaluated considering the auxiliary operator $\mathcal{O} = (c_{\alpha\sigma} + c_{\beta\sigma})$ and using the relation $G_{\alpha\beta\sigma\sigma} = \frac{1}{2}(C_{\mathcal{O}} - G_{\alpha\alpha\sigma\sigma} - G_{\beta\beta\sigma\sigma})$.

The diagonal spin, charge and pair susceptibility terms $\chi^{S^z}_{\alpha\alpha}$, $\chi^{N}_{\alpha\alpha}$ and $\chi^{\Delta}_{\alpha\alpha}$ are evaluated employing, respectively, the operators $\mathcal{A} = \sum_{\sigma\sigma'} c^+_{\alpha\sigma}\tau^z_{\sigma\sigma'}c_{\alpha\sigma'} \equiv S^z_\alpha$, $\mathcal{A} = \sum_{\sigma\sigma'} c^+_{\alpha\sigma}\tau^0_{\sigma\sigma'}c_{\alpha\sigma'} \equiv N_\alpha$) and $\mathcal{A} = c_{\alpha\downarrow}c_{\alpha\uparrow} \equiv \Delta_\alpha$, where $\tau^{a=0,x,y,z}$ are the Pauli matrices. Similarly, yhe off-diagonal terms are obtained by considering the operator $\mathcal{A} = S^z_\alpha + S^z_\beta$, $\mathcal{A} = N_\alpha + N_\beta$ or $\mathcal{A} = \Delta_\alpha + \Delta_\beta$. The excitonic susceptibility $\chi^T_{\alpha\beta}$ is defined with respect to the vector operator: $T^i_{\alpha\beta} = \sum_{\sigma\sigma'} c^+_{\alpha\sigma}\tau^i_{\sigma\sigma'}c_{\beta\sigma'}$, respectively, the spin-singlet ($i = 0$) and the spin-triplet excitons ($i = x, y, z$) operators.

superc. In the superconductive case the Nambu orbital dependent s-wave Green's function reads:

$$
\hat{G}_{\alpha\beta} = \begin{bmatrix} G_{\alpha\beta\uparrow\uparrow} & F_{\alpha\beta\uparrow\downarrow} \\ \bar{F}_{\alpha\beta\downarrow\uparrow} & \bar{G}_{\alpha\beta\downarrow\downarrow} \end{bmatrix}
\qquad \texttt{GFnambu} \quad (8)
$$

However, leveraging on the symmetries holding between the different components we can reduce to evaluate only the top row components plus auxiliary functions. The calculation of the diagonal normal component $G_{\alpha\alpha\uparrow\uparrow}$ follows the very same scheme of the **normal** case. For the off-diagonal terms however we do not rely on any symmetry relation but we consider two auxiliary functions $C_{\mathcal{O}}$, $C_{\mathcal{P}}$ with $\mathcal{O} = c_{\alpha\uparrow} + c_{\beta\uparrow}$, $\mathcal{P} = c_{\alpha\uparrow} - ic_{\beta\uparrow}$ so that: $G_{\alpha\beta\uparrow\uparrow} = \frac{1}{2}[C_{\mathcal{O}} + C_{\mathcal{P}} - (1-i)(G_{\alpha\alpha\uparrow\uparrow} + G_{\beta\beta\uparrow\uparrow})$. The diagonal and off-diagonal anomalous terms $F_{\alpha\beta\uparrow\downarrow}$ need to be evaluated using different combinations of creation/destruction operators. First we evaluate the $\bar{G}_{\alpha\alpha\downarrow\downarrow}$ component as an auxiliary term considering $\mathcal{A} = c^{+}_{\alpha\downarrow}$. Next, we consider the two linear combinations: $\mathcal{T} = c_{\alpha\uparrow} + c^{+}_{\beta\downarrow}$, $\mathcal{R} = c_{\alpha\uparrow} - ic^{+}_{\beta\downarrow}$ contributing, respectively, to the auxiliary functions $C_{\mathcal{T}}$, $C_{\mathcal{R}}$. The searched anomalous functions reads: $F_{\alpha\beta\uparrow\downarrow} = \frac{1}{2}[C_{\mathcal{T}} + C_{\mathcal{R}} - (1-i)(G_{\alpha\alpha\uparrow\uparrow} + \bar{G}_{\beta\beta\downarrow\downarrow})]$.

nonsu2. In this case all the Green's functions components need to be evaluated. For the orbital and spin diagonal terms $G_{\alpha\alpha\sigma\sigma}$ we proceeds as in the **normal** case. The remaining off-diagonal components $G_{\alpha\beta\sigma\sigma'}$ are evaluated using auxiliary operators $\mathcal{O} = c_{\alpha\sigma} + c_{\beta\sigma'}$, $\mathcal{P} = c_{\alpha\sigma} - ic_{\beta\sigma'}$ and the relation: $G_{\alpha\beta\sigma\sigma'} = \frac{1}{2}[C_{\mathcal{O}} + C_{\mathcal{P}} - (1-i)(G_{\alpha\alpha\sigma\sigma} + G_{\beta\beta\sigma'\sigma'})]$.

3.8. Observables

`sSecObc`

A number of predefined impurity observable or local static correlations, e.g. occupation, total energy, pair amplitude, excitonic order parameter ,etc. are calculated in the module `OBSERVABLES`. As for the previous cases, this module wraps the different implementations and quantities to evaluate from the operational modes `ed_mode=?` implemented in `ED_OBSERVABLES_?`. Local observables and correlations are defined by the thermal average $\langle\mathcal{O}\rangle = \mathrm{Tr}\left[e^{-\beta H}\mathcal{O}\right]/Z$. This can be efficiently evaluated at zero and low temperature using the stored low energy part of the spectrum leveraging on the exponential cutoff from the Boltzmann factor:

$$\langle\mathcal{O}\rangle = \frac{1}{Z}\sum_{n} e^{-\beta E_n}\langle\psi_n|\mathcal{O}|\psi_n\rangle$$

where E_n and $|\psi_n\rangle$ are the low lying eigenstates of the system stored in the `state_list`.

3.9. Reduced impurity density matrix

`sSecRDM`

This version of the EDIpack2 library features the calculation of the impurity Reduced Density Matrix (iRDM), enabling the analysis of entanglement properties of the quantum impurity for any value of `ed_mode`. To simplify the discussion in this section we take the zero temperature limit and consider the presence of a non-degenerate groundstate $|\psi\rangle$ populating the `state_list`. The generalization to the presence of degenerate states or the finite temperature regime are straightforward. The groundstate belongs to a given symmetry sector $\mathcal{S}$ and it represented in the Fock basis as: $|\psi\rangle = \sum_I a_I|I\rangle$. The density matrix ρ is defined as:

$$\rho = |\psi\rangle\langle\psi| = \sum_{nm=1}^{4^{N_s}} a^*_m a_n |n\rangle\langle m| = \sum_{nm=1}^{4^{N_s}} \rho_{nm}|n\rangle\langle m|$$

The iRDM is obtained from this by tracing out the bath degrees of freedom: $\rho^i = Tr_b\rho$. The summation appearing in the previous expressions becomes quickly unmaneagable with the system size. To overcome this issue we implemented a fast algorithm leveraging on symmetry sector and map sparsity. Using quantum number conservation the summation extrema get reducedto the size D_S of the sector S to which the groundstate (or in general any eigenstate) belongs. Next we analyze the groundstate expansion on the Fock states, splitting the bit representation of any state $|I\rangle$ into their spin-dependent impurity and bath parts as $|I\rangle = |i_\uparrow\, b_\uparrow\, i_\downarrow\, b_\downarrow\rangle$. Each bit set span a space of dimension D^{α}_{imp} or D^{σ}_{bath}. Finally, we store into the `sparse_map` $\mathcal{S}$, for every impurity configuration $i_\sigma = 1,\dots,D^{\sigma}_{imp}$ (key) the corresponding bath states $j \in [1,\dots,D^{\sigma}_{bath}]$ (values) which contribute to that key, as detailed in Sec. 3.4.2. We indicate with $\mathcal{D}^{\sigma}_{i}$ the number of values for the key i_σ. The fast summation algorithm for the evaluation of the iRDM takes on a different form for the two cases **normal** and **superc/nonsu2**, as outlined below.

normal. In this case we take advantage of the Fock and sector spin resolved factorization discussed in Sec. 3.3. As such we can also split impurity and bath bits independently for each spin configuration:

$$|\vec{n}\rangle = |\vec{n}_\uparrow\rangle \otimes |\vec{n}_\downarrow\rangle = |\vec{i}_\uparrow\rangle|\vec{b}_\uparrow\rangle \otimes |\vec{i}_\downarrow\rangle|\vec{b}_\downarrow\rangle$$

In addition it should be noted that creation (destruction) operators at position p for spin σ $c^+_{p\sigma}$ ($c^+_{p\sigma}$) do not operate on the opposite spin so that they pass through without taking any fermionic sign. Finally, we observe that local iRDM admits only diagonal terms in the **normal** case. Thus we obtain for the iRDM:

$$\begin{aligned}
\rho^i &= \sum_{\sigma=\uparrow\downarrow}\sum_{b_\sigma=1}^{D_{b_\sigma}} \langle b_\uparrow| \otimes \langle b_\downarrow|\rho|b_\downarrow\rangle \otimes |b_\uparrow\rangle \\
&= \sum_{\sigma=\uparrow\downarrow}\sum_{b_\sigma=1}^{D_{b_\sigma}}\sum_{i_\sigma=1}^{D_{i_\sigma}}\sum_{p_\sigma=1}^{D_{p_\sigma}}\sum_{j_\sigma=1}^{D_{j_\sigma}}\sum_{q_\sigma=1}^{D_{q_\sigma}} C_{i,p}\, C_{j,q}\, a_{i_\uparrow p_\uparrow i_\downarrow p_\downarrow} a^*_{j_\uparrow q_\uparrow j_\downarrow q_\downarrow} \langle b_\uparrow|p_\uparrow\rangle|i_\uparrow\rangle\langle j_\uparrow|\langle q_\uparrow|b_\uparrow\rangle \otimes \langle b_\downarrow|p_\downarrow\rangle|i_\downarrow\rangle\langle j_\downarrow|\langle q_\downarrow|b_\downarrow\rangle \\
&= \sum_{b_\sigma=1}^{D_{b_\sigma}}\sum_{i_\sigma=1}^{D_{i_\sigma}} a_{i_\uparrow b_\uparrow i_\downarrow b_\downarrow} a^*_{i'_\uparrow b_\uparrow i'_\downarrow b_\downarrow} |i_\uparrow\rangle\langle i'_\uparrow| \otimes |i_\downarrow\rangle\langle i'_\downarrow|
\end{aligned} \tag{9}$$

iRDMnormal

where the constant $C_{i,b}$ take care of the sign change due to exchanging bath and impurity bit sets. As effect of the Kronecker delta symbols for bath and impurity these constants are identically 1. The numerical implementation relies on the use of the `sparse_map` ($\%sp$) for the groundstate sector as reported in the listing below:

```
do IimpUp=0,2**Norb-1
  do JimpUp=0,2**Norb-1
    !Finding the unique bath states connecting IimpUp and JimpUp
    call sp_return_intersection(sectorI%H(1)%sp,IimpUp,JimpUp,BATHup,lenBATHup)
    if(lenBATHup==0)cycle
    do IimpDw=0,2**Norb-1
      do JimpDw=0,2**Norb-1
        !Finding the unique bath states connecting IimpDw and JimpDw -> BATHdw(:)
        call sp_return_intersection(sectorI%H(2)%sp,IimpDw,JimpDw,BATHdw,lenBATHdw)
        if(lenBATHdw==0)cycle
        do ibUP=1,lenBATHup
          IbathUp = BATHup(ibUP)
          do ibDW=1,lenBATHdw
            IbathDw = BATHdw(ibDW)
            !Allowed spin Fock space Istates:
            !Iup = IimpUp +  2^Norb * IbathUp
            !Idw = IimpDw +  2^Norb * IbathDw
            iUP= binary_search(sectorI%H(1)%map,IimpUp + 2**Norb*IbathUp)
            iDW= binary_search(sectorI%H(2)%map,IimpDw + 2**Norb*IbathDw)
            i  = iUP + (iDW-1)*sectorI%DimUp
            !Allowed spin Fock space Jstates:
            !Jup = JimpUp +  2^Norb * IbathUp
            !Jdw = JimpDw +  2^Norb * IbathDw
            jUP= binary_search(sectorI%H(1)%map,JimpUp + 2**Norb*IbathUp)
            jDW= binary_search(sectorI%H(2)%map,JimpDw + 2**Norb*IbathDw)
            j  = jUP + (jDW-1)*sectorI%DimUp
            !
            io = (IimpUp + 2**Norb*IimpDw) + 1
            jo = (JimpUp + 2**Norb*JimpDw) + 1
            irdm(io,jo) = irdm (io,jo) + psi(i)*psi(j)*weight
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
```

superc/nonsu2. In presence of these symmetries suitable modifications related to the absence of Fock space factorization have to be taken into account. To start with a generic Fock state appearing in the groundstate decomposition has the form:

$$|\vec{n}\rangle = |\vec{i}_\uparrow\vec{b}_\uparrow\vec{i}_\downarrow\vec{b}_\downarrow\rangle$$

Inserting this latter in the definition of iRDM we obtain:

$$\begin{aligned}\mathrm{Tr}_{b_\uparrow,b_\downarrow}\rho &= \sum_{b_\sigma=1}^{D_{b_\sigma}} \langle b_\uparrow\, b_\downarrow|\rho|b_\downarrow\, b_\uparrow\rangle \\ &= \sum_{b_\sigma=1}^{D_{b_\sigma}}\sum_{i_\sigma=1}^{D_{i_\sigma}}\sum_{p_\sigma=1}^{D_{p_\sigma}}\sum_{j_\sigma=1}^{D_{j_\sigma}}\sum_{q_\sigma=1}^{D_{q_\sigma}} a_{i_\uparrow p_\uparrow i_\downarrow p_\downarrow} a^*_{j_\uparrow q_\uparrow j_\downarrow q_\downarrow} \langle b_\uparrow\, b_\downarrow|i_\uparrow\, p_\uparrow\, i_\downarrow\, p_\downarrow\rangle\langle q_\downarrow\, j_\downarrow\, q_\uparrow\, j_\uparrow|b_\downarrow\, b_\uparrow\rangle\,.\end{aligned} \tag{10}$$

In order to simplify this expression we should get rid of the sum with respect to the bath indices p_σ and q_σ, ideally contracting them with the outer indices b_σ. However, this requires to bring all the bath bits so to preced the impurity ones:

$$|i_\uparrow\, p_\uparrow\, i_\downarrow\, p_\downarrow\rangle \to C_{p_\uparrow,i_\downarrow}|i_\uparrow\, i_\downarrow\, p_\uparrow\, p_\downarrow\rangle$$

where $C_{p_\uparrow,i_\downarrow} = (-1)^{\#\vec{n}_{p\uparrow}\cdot\#\vec{n}_{i\downarrow}}$ with $\#\vec{n} = \sum_i n_i$ is an overall fermionic sign which depends on the occupation of the $p_\uparrow$ and the $i_\downarrow$ bit configurations. Inserting this relation in the equation for the iRDM we obtain:

$$\begin{aligned}\mathrm{Tr}_{b_\uparrow,b_\downarrow}\rho &= \sum_{i_\sigma=1}^{D_{i_\sigma}}\sum_{j_\sigma=1}^{D_{j_\sigma}}\left[\sum_{b_\sigma=1}^{D_{b_\sigma}} a_{i_\uparrow b_\uparrow i_\downarrow b_\downarrow} a^*_{j_\uparrow b_\uparrow j_\downarrow b_\downarrow} C_{b_\uparrow,i_\downarrow} C_{b_\uparrow,j_\downarrow}\right]|i_\uparrow\, i_\downarrow\rangle\langle j_\downarrow\, j_\uparrow| \\ &= \sum_{i_\sigma=1}^{D_{i_\sigma}}\sum_{j_\sigma=1}^{D_{j_\sigma}} \rho^i_{i_\uparrow i_\downarrow j_\downarrow j_\uparrow}|i_\uparrow\, i_\downarrow\rangle\langle j_\downarrow\, j_\uparrow|\end{aligned} \tag{11}$$

The numerical implementation requires few modifications with respect to the previous case. In particular it is important to recall that the ordering of the key-value pairs in the `sparse_map` in this case follow a contiguous ordering and that their decomposition is necessary to reconstruct the correct Fock state index, see the following listing:

```
do IimpUp=0,2**Norb-1
  do IimpDw=0,2**Norb-1
    do JimpUp=0,2**Norb-1
      do JimpDw=0,2**Norb-1
        !Build indices of the RDM in 1:4**Norb
        iImp = iImpUp + iImpDw*2**Norb
        jImp = jImpUp + jImpDw*2**Norb
        call sp_return_intersection(sectorI%H(1)%sp,iImp,jImp,Bath,lenBath)
        if(lenBATH==0)cycle
        do ib=1,lenBath
          iBath = Bath(ib)
          !Decompose iBath into B_Up, B_dw to reconstruct B bit set.
          !B --> {B_up,B_dw}
          iBathUp = mod(iBath,2**Nbath)
          iBathDw = (iBath)/2**Nbath
          !Reconstruct the Fock state Ii map back to sector state i
          ii= iImpUp + iImpDw*2**Ns + 2**Norb*(IBathUp + IBathDw*2**Ns)
          i = binary_search(sectorI%H(1)%map,ii)
          !Reconstruct the Fock state Jj map back to sector state j
          jj= jImpUp + jImpDw*2**Ns + 2**Norb*(IBathUp + IBathDw*2**Ns)
          j = binary_search(sectorI%H(1)%map,jj)
          !Build the signs of each components of RDM(io,jo)
          nBup  = popcnt(Ibits(ii,Norb,Norb*Nbath))
          nIdw  = popcnt(Ibits(ii,Ns,Norb))
          nJdw  = popcnt(Ibits(jj,Ns,Norb))
          signI = (-1)**(nIdw*nBup)
          signJ = (-1)**(nJdw*nBup)
          sgn   = signI*signJ
          !
          io = (iImpUp+1) + 2**Norb*iImpDw
          jo = (jImpUp+1) + 2**Norb*jImpDw
          rdm(io,jo) = rdm(io,jo) + psi(i)*conjg(psi(j))*weight*sgn
        enddo
      enddo
    enddo
  enddo
enddo
```

3.10. Bath Functions

sSecFunc

The module `ED_BATH_FUNCTIONS` implements on-the-fly calculations of the hybridization function $\Delta_{\alpha\beta\sigma\sigma'}(z) = \sum_\nu V^\nu_{\alpha\sigma}(z\mathbb{1}-h^\nu_{\alpha\beta\sigma\sigma'})^{-1}V^\nu_{\beta\sigma'}$ and the non-interacting Green's functions $G^0_{\alpha\beta\sigma\sigma'}(z) = [(z+\mu)\delta_{\alpha\beta\sigma\sigma'} - h^0_{\alpha\beta\sigma\sigma'} - \Delta_{\alpha\beta\sigma\sigma'}(z)]^{-1}$ for any frequency $z \in \mathbb{C}$. The module contains implementations for any case according to the values of `ed_mode` and `bath_type`, either with supplied user bath (a rank-1 array of doubles) or using the allocated `effective_bath` instance `dmft_bath`. We provide different functions to get either Δ, G^0 and its directly its inverse $[G^0]^{-1}$, which are used to evaluate the self-energy functions upon request, see Sec. 3.12. In the superconductive case `ed_mode`=**superc** different procedures can be used to retrieve the anomalous components of any function defined above in the Nambu basis.

3.11. Bath Optimization

sSecFit

to be revised, copy&paste from edipack paper: well it is essentiall the same shit In the DMFT context the bath needs to be optimized against a given realization of the Weiss field ${\mathcal{G}_0^{-1}}_{\alpha\beta\sigma\sigma'}(z)$, or the corresponding *hybridization* function $\Theta = (z+\mu)\mathbb{1} - H^{loc} - \mathcal{G}_0^{-1}](z)$. While the Weiss field is obtained from the DMFT self-consistency equation[6], bath discretization impose to find an optimal description of the continuous Weiss field in terms of the finite number of parameters of the bath (see Sec. 3.5. Although different algorithms can be envisaged to perform this step[7,8] which are perfectly valid in some physical contexts. In order to enable the users to use their preferred method in EDIpack2 we keep the optimization implementation independent from the rest of the code. Yet, we offer a fully fledged optimization strategy based on the conjugate gradient (CG) minimization of the cost function:

$$\chi = \sum_{n=1}^{L_{fit}} \frac{1}{W_n} ||X(i\omega_n) - X^{QIM}(i\omega_n;\{V,h\})||_q$$

where the symbol $||\cdot||_q$ is a suitably defined q-norm in the matrix function space, $X_{\alpha\beta} = \mathcal{G}_{0\alpha\beta\sigma\sigma'}$, $\Theta_{\alpha\beta\sigma\sigma'}$ are the user supplied local functions and $X^{QIM}_{\alpha\beta} = G^0_{\alpha\beta\sigma\sigma'}$, $\Delta_{\alpha\beta\sigma\sigma'}$ the quantum impurity model functions, defined above. Notably, more sophisticated optimization algorithms have been recently implemented to better undertake the minimization of similar cost functions. However, these developments are designed to work with truncated algorithms, e.g. sCI or DMRG, describing systems with a larger number of bath levels. In the ED context however, where the efficient and wise selection of the relatively few bath levels becomes crucial, the CG minimization of the local functions proved to be the most efficient and flexible optimization scheme so far.

The fit procedure is entirely wrapped by the single function `ed_chi2_fitgf` contained in the module `ED_BATH_FIT`. In order to exploit regularity of the functions the fit is performed using imaginary Matsubara frequency. The actual form of $X_{\alpha\beta\sigma\sigma'}$ is controlled by the input parameter `cg_Scheme=Weiss,Delta`.

In order to fine tune the optimization step in EDIpack2 we introduced a number of control parameters. To start with, we include two distinct CG algorithms in the library, controlled by the input variable `cg_method=0,1`. The value `0` corresponds to a Fletcher-Reeves-Polak-Ribiere minimisation algorithm, adapted from Numerical Recipes[9]. Yet, in order to guarantee back-compatibility with respect to the past literature, we also provide the possibility of using the minimization procedure published in Ref. 6 which has been largely used in the DMFT community.

The gradient $\nabla\chi$ required by the CG algorithm can be evaluated either analytically or numerically as controlled by the value of the input parameter `cg_grad=0,1`. For `cg_method=1` only numerical gradient can be used. The number of Matsubara frequency L_{fit} used in the fit procedure is controlled by the input parameter `cg_Lfit`. This parameter can be used to restrict the fit to the low frequency part as the large frequency tails of the local Matsubara functions have universal behavior. Similarly, the weight $W_n = 1, 1/L_{fit}, 1/\omega_n$ is used to enhance the weight of low energy part with respect to the intermediate-to-large energy one in the cost function. This is controlled by the input parameter `cg_Weight=0,1,2`. Another factor contributing to determine the behavior of the fit is the power q of the cost function χ. By tuning this parameter, controlled by input variable `cg_pow`, it is possible to enhance or suppress the differences in the χ in order to improve (or simplify) the optimization procedure.

Finally, we introduced few additional parameters which aim to control the quality of the fit. The first is the fit tolerance, controlled by the input parameter `cg_Ftol`, which determines the convergence threshold of the minimization procedure. The second is the maximum number of iterations `cg_Niter` allowed for the CG algorithm to determine the minimum. The last parameter `cg_stop=0,1,2` selects the exit condition of the minimization procedure. We envisaged three possible conditions: $C_1 \cup C_2$, C_1, C_2 with $C_1 = |\chi^{n-1} - \chi^n| < \texttt{cg\_Ftol}(1 + \chi^n)$ and $C_2 = ||x_{n-1} - x_n|| < \texttt{cg\_Ftol}(1 + ||x_n||)$, where χ^n is the cost function evaluated at the n^{th}-step and x_n the corresponding argument. Increasing the value of the stop parameter generally loosens up the exit conditions of the minimization.

REMARK TODO Notes on the norm function: a) the definition for conventional matrices, ;b) the definition for the replica/general case where you can choose between different norm defition.

3.12. Input/Output

`sSecIO`

The module `ED_IO` provides access to the results of the Lanczos diagonalization of the quantum impurity problem. As any single instance of the code is preserved in the memory until a new call is made, access to the memory pool is only possible through specific functions implemented in `ED_IO`. For instance, dynamical response functions, self-energy components or impurity observables can all be retrieved using dedicated functionalities. Additionally we provide procedures to perform on-the-fly re-calculation of the impurity Green's functions and self-energy on a given arbitrary set of points in the complex frequency domain exploiting their storage in terms of `gfmatrix` objects. A long list of available functions is contained in the on-line documentation (see https://edipack.github.io/EDIpack2.0/edipack2/13_io.html#ed-io) Here we limit the discussion to two paradigmatic examples. The first is the function `ed_get_dens` which write on the provided input array the value of the orbital impurity occupations $\langle n_{\alpha\sigma} \rangle$. Analogously `ed_get_sigma` is used to retrieve the normal or anomaous components of the Matsubara or Real-axis self-energy function $\Sigma(z)$. This latter represents the key output of the calculation in any DMFT application.

3.13. EDIpack2ineq: inequivalent impurities

`sSecIneq`

It is often necessary to solve systems with several, independent, quantum impurities. Paradigmatic examples arise in the DMFT framework when dealing with the effective description of lattices built out of unit cells hosting multiple inequivalent atoms or large super-cell systems with possibly broken translational symmetries, e.g. heterostructures of correlated materials, disordered systems, etc. As the structure of EDIpack2 allows the presence of a single instance of the solver at any time, we implemented a suitable extension to deal with multiple impurities. Specifically `EDIpack2Ineq` is a sub-library which complements EDIpack2 , wrapping memory and procedures in order to enable simultaneous treatment of inequivalent impurities. Using standard Fortran interfaces it is possible to group all the EDIpack2 and `EDIpack2ineq` functions under the same name without compromising the software functionalities. Here below we briefly discuss the main additions introduced by `EDIpack2ineq`.

3.13.1. Structure

`ssSecIneqStructure`

The sub-library is composed of a number of Fortran modules which are wrapped by `EDIPACK2INEQ`. This latter represents the main user interface and gives access to all the available procedures and variables as needed to solve quantum impurity problems. The user needs to invoke use of this module alongside the EDIpack2 one:

```
program test
  !Load EDIpack2 library
  USE EDIPACK2
  !Load the Inequivalent impurities extension
  USE EDIPACK2INEQ
  ...
```

A major part of the `EDIpack2ineq` library is to define global variables which wrap and conserve the memory pool of EDIpack2 . For instance we implement a number of globally shared arrays with an increased rank to store the variables in any memory instance of EDIpack2 and accessible through the input/output procedures (see Sec. 3.12). Specific objects, e.g. `effective_bath` or `gfmatrix`, in principle require dedicated

MPI communication when parallel execution of the solver among independent impurities. However, we opt for the much simpler and economic solution of reading these objects from the file in which they get saved anyway.

3.13.2. Core routines `ssSecIneqGlobal`

The module `E2I_MAIN` wraps the extensions to the main algorithms of EDIpack2 , namely: initialization, diagonalization and finalization, preserving the original naming convention. The initialization process is handled by `ed_init` which is extended to accepts a bath array or rank-2 (as opposed to rank-1 array in the EDIpack2 implementation). The leading dimension now determines the number of inequivalent impurity problems to be solved. This procedure initialize all the `EDIpack2ineq` variables and prepare each bath for the diagonalization step. In `ed_solve`, which accepts the same rank-2 bath array, performs the diagonalization of each quantum impurity problem in either serial of parallel mode. The parallel mode is further controlled by the input function `mpi_lanc=T/F`. If true Lanczos diagonalization is performed in parallel and inequivalent sites are treated sequentially. The inverse if false. The finalization procedure with full memory release is performed in `ed_finalize`.

3.13.3. Inequivalent Baths `ssSecIneqBath`

The inequivalent impurities extension is encoded in `E2I_BATH` which contains procedures to control the baths setup on the user side and the implementation of conventional symmetry operations. In the `E2I_BATH_REPLICA` module we extend all the functions which set the replica/general bath matrix basis $\vec{\Gamma}$ and the corresponding initial variational parameters $\vec{\lambda}$. Although the variational parameters are generally different from site to site, in the current implementation the matrix basis is independent from the impurities number. This will possibly be relaxed in a future release. Analogously, in the module `E2I_BATH_FIT` we implement an extension of the generic function `ed_chi2_fitgf` to perform multiple independent bath optimization for all inequivalent impurities in terms of an MPI iterative cycle.

3.13.4. Input/Output `ssSecIneqIO`

A key part of the `EDIpack2ineq` library is the extension for the input/output procedures to return observables or dynamical functions per inequivalent impurity. The module `E2I_IO` implements numerous functions which retrieve or evaluate on-the-fly any quantity accessible in EDIpack2 for any given impurity (or all of them) using the same naming original convention defined in EDIpack2 .

4. Interoperability

`SecInterop`

The recent growing availability of state-of-the-art software dedicated to the solution of quantum impurity problems using different methods poses a serious challenge to test accuracy and realiability of the results. As such, complex softwares are requested to develop a higher level of interoperability, i.e. the possibility to operate with other software possibly using different programming language. Modern Fortran, which is the language of choice for EDIpack2 , since many years supports the standardized generation of procedures and variables that are interoperable with C.

4.1. C-bindings `sSecInteropCbindings`

The interoperability with C language is provided by the `ISO_C_BINDING` module, which is part of the Fortran standard since 2003. The module contains definitions of named constants, types and procedures for C interoperability. Alongside, a second key feature essential to expose to C any Fortran entity is the `BIND(C)` intrinsic function. In EDIpack2 we exploit these features of the language to provide a complete interface from the Fortran code to C/C++, i.e. `EDIpack2C`.

4.1.1. Installation

sSecInteropCbindingsInstallation

The C-binding module in included in the build process of EDIpack2 and compiled into a dynamical library `libedipack2_cbinding.so/.dylib`. As dicussed in the Sec. 2.3, the support for inequivalent impurities is configured at the build level and propagates to the C-bindings library as well. An exported variable `has_ineq` is defined and export to C/C++ as a way to query the presence of inequivalent impurities support. The generated library and header files get installed in the include directory at the prefix location, specified during configuration step. The corresponding path is added to the environment variable `LD_LIBRARY_PATH`, valid of any Unix/Linux system, via any of the loading methods outlines in Sec. 2.4.

4.1.2. Implementation

sSecInteropCbindingsImplementation

The interface layer is contained in the Fortran module `EDIpack2_C`. This contains a common part and two set of functions, one to interface the procedures from EDIpack2 and a second one to extend the interface to the inequivalent impurities case. Specifically, the implemented interface functions expose to C through `bind(C)` statement a number of procedures composing the Fortran API of EDIpack2 , i.e. contained in `ED_MAIN`. The procedures and shared variables can be divided in four main groups:

Variables. A number of relevant input and shared variables which are normally required to setup or to control the calculation are interfaced to C directly in the EDIpack2 modules `ED_INPUT_VARIABLES` using `bind(C)` constructs. These are implicitly loaded into the C-binding module `EDIpack2_C` through the Fortran `USE EDIPACK2` statement and then further interfaced in the C++ header file.

Main. This group contains interface to the exact digonalization solver, interfacing the solver itself `solve_site/ineq` as well as its initialization `init_solver_site/ineq` and finalization `finalize_solver` procedures. It also includes the functions used to set the non-interacting part of the impurity Hamiltonian $h^0_{\alpha\beta\sigma\sigma'}$ through the functions `set_Hloc`.

Bath. In this group we implement a number of procedures dealing with bath initialization, symmetry operations and optimization. In particular it contains the function returning the dimension of the bath array on the user side `get_bath_dimension` as well as the setup of the matrix basis $\vec{\Gamma}$ for the replica/general bath via different instances of `set_Hreplica/general_site,lattice_d3,5`. The crucial part in a DMFT framework is the optimization of the bath through conjugate-gradient algorithm. To this end we interfaced a number of functions `chi2_fitgf_single,lattice_normal,superc_n3,4,5,6` which cover all the cases supported in EDIpack2 . Note that, because the actual optimization is still performed through the Fortran code, no changes apply to the outcome of this step.

Input/Output. The input and output part of the software is interfaced in this group of functions. In particular, `read_input` expose to C the input reading procedure of EDIpack2 which set essentially set all the internal variables of EDIpack2 . Next, in `edipack2ineq_c_binding_io` we interface all the functions implementing the communication from the EDIpack2 instance to the user, namely those to retrieve static observables (e.g. `get_dens`), the impurity Green's functions and self-energies (e.g. `get_Sigma_site,lattice_n3,5`), the impurity susceptibilities (e.g. `get_spinChi`), the impurity reduced density matrix as well as the non-interacting Green's and hybridization functions starting from the user bath array.

Finally, in order to ensure compatibility to C and C++ language the entire set of C-binding functions and variables is enclosed into a header file `edipack2_cbinding.h`.

4.2. EDIpy2, the Python API

sSecInteropEDIpy

As a first application of the EDIpack2 C-bindings we implemented a complete Python interface, i.e. EDIpy2. This is a Python module which enables access to all the library features and unlock implementation of further interfaces of EDIpack2 as a plug-in solver for external Python based software.

4.2.1. Installation `sSecInteropEDIpyInstallation`

EDIpy2 is available as a stand-alone module which only depends on EDIpack2 and SciFortran. The Python package can be obtained from the repository EDIpy2.

```
git clone https://github.com/edipack/EDIpy2 EDIpy2
cd EDIpy2
pip install .
```

In some more recent Python distribution the flag `--break-system-packages` might be required to complete installation or a virtual environment should be used instead.

PyPi. The EDIpy2 package is also available in PyPi at pypi.org/edipy2. As such EDIpy2 can be conveniently installed in any system supporting `pip` as:

```
pip install edipy2
```

Anaconda. As for EDIpack2 , the Python API in EDIpy2 is available through Anaconda packaging for Unix/Linux systems. Packages are available for Python version > 3.10. The EDIpack2.0 package contains the `edipy2` Python module as well as the EDIpack2.0 and SciFortran libraries. In this case the resolution of the dependencies is taken care from Conda itself. Using conda the installation reads:

```
conda create -n edipack
conda activate edipack
conda install -c conda-forge -c edipack edipack2
```

Once installed, any loading of the `edipy2` module resolves in the attempt to load the dynamic library `libedipack2_cbinding.so/.dylib` containing the Fortran-C bindings for EDIpack2 . By default the library is searched proceeds as follow:

1. First the user can override the location of the library (determined during EDIpack2 build configuration) by exporting an environment variable called `EDIPACK_PATH` .
2. By default, the Python module detects the position of the Fortran libraries via `pkg-config`. Any of the OS loading method outlined in Sec. 2.4 automatically push the correct configuration to the `PKG_CONFIG_PATH`.
3. As a last resort the environment variables `LD_LIBRARY_PATH` and `DYLD_LIBRARY_PATH` are analyzed to retrieve the correct position.

If none of the previous attempts succeeds, the module will not not load correctly and an error message will be printed.

4.2.2. Implementation `sSecInteropEDIpyImplementation`

The Python API provided in the edipy2 module consists essentially of a suitable class called for convenience `global_env`. The class incorporates all the global variables inherited from EDIpack2 C-bindings library and implements a number of interface functions leveraging the Python duck typing to EDIpack2 . The variables and the functions of EDIpack2 are exposed to the user and are accessed as properties and methods of the `global_env` claass, which needs to be imported at the beginning of the python script, along with other useful modules. For instance Numpy is necessary, while mpi4py is strongly recommended.

```
import numpy as np
import mpi4py
from mpi4py import MPI
from edipy2 import global_env as ed
import os,sys
```

The EDIpy2 supports the solution of problems with independent impurities, interfacing with the the EDIpack2ineq extension of the library, if present. Should the inequivalent impurities package not be built, the Python module silently disable the support to it, so that invoking any related procedure will result in a `RuntimeError`. The user can check the availability of the inequivalent impurities interface by querying the value of `edipy2.global_env.has_ineq`.

The implementation of the Python API is divided in two main parts. The first is a set of global variables, the second includes 4 groups of functions: solver, bath, input/output, auxiliary.

Global variables. This includes a subset of the input variables available in EDIpack2 which are used to control the calculation. The variables are loaded globally in `EDIpy2` and can be accessed or set locally as properties of the class `global_env`. Differently these are initialized, alongside the remaining default input variables, through a call to procedure `edipy2.global_env.read_input` which interface the `ed_read_input` function in EDIpack2 . A given example is reported in the following code extract:

```
import numpy as np
from edipy2 import global_env as ed
ed.Nspin = 1              # set a global variable
mylocalvar = ed.Nspin     # assign to a local variable
print(ed.Nspin)           # all functions can have global variables as arguments
np.arange(ed.Nspin)
```

Solver functions. This group includes a number of functions enabling initialization, execution and finalization functionalities for the diagonalization solver.

- `init_solver` and `set_Hloc`. The first initializes the EDIpack2 environment for the quantum impurity problem solution, sets the effective bath either reading it from a file or initializing it from flat band model. Once this function is called, it is not possible to allocate a second instance of the solver. `set_Hloc` sets the non-interacting part of the impurity Hamiltonian $\eta^0_{\alpha\beta\sigma\sigma'}$. Either function take different argument combinations there including support for inequivalent impurities.

- `solve` This function solves the quantum impurity problem, calculates the observables and any dynamical correlation function. All results remain stored in the memory and get accessed through input/output functions.

- `finalize_solver` This function cleans up the EDIpack2 environment, free the memory deallocating all relevant arrays or data structure. A call to this functions enables a new initialization of the solver, i.e. a new call to `init_solver`.

Bath functions. This set covers the implementation of utility functions handling the effective bath on the user side as well as interfaces to specific EDIpack2 procedures, either setting bath properties or applying conventional symmetry transformation. Here we discuss a pair of crucial functions in this group.

- `bath_inspect`This function translates between the user-accessible continuous bath array and the bath components (energy level, hybridization and so on). It functions in both ways, given the array returns the components and vice-versa. It autonomously determines the type of bath and ED mode.

- `chi2_fitgf` This function fits the Weiss field or Hybridization function (delta) with a discrete set of level. The fit parameters are the bath parameters contained in the user-accessible array. Depending on the type of system we are considering (normal, superconductive, non-SU(2)) a different set of inputs has to be passed. The specifics of the numerical fitting routines are controlled in the input file.

Additionally, the group includes the function `get_bath_dimension`, returning the correct dimension for the user bath array to be allocated and `set_Hreplica/general` which set the matrix basis $\vec{\Gamma}$ and init the bath variational parameters $\vec{\lambda}$ for `bath_type=replica,general`.

Input/Output functions. This group includes functions which return to the userspace observables or dynamical correlation functions evaluated in EDIpack2 and conserved in the corresponding memory instance. Each function provides a general interface which encompass all dimension of the input array there including inequivalent impurities support. For instance, the function `get_sigma` returns the self-energy function array (evaluated on-the-fly) for a specified supported shape, normal or anomlous type and on a specific axis or frequency domain.

Auxiliary functions. This group include some auxiliary functions, either interfacing EDIpack2 procedures or defined locally in Python to provide specific new functionalities. Among the latter we include `get_ed_mode` which return the value of the variable `ed_mode=normal,superc,nosu2` and `get_bath_type` which similarly returns the value of `bath_type`.

4.3. TRIQS interface

`sSecInteropTRIQS`

Igor A purely Python EDIpack2 to Triqs interface is available, leveraging on the C-bindings and Python API. The corresponding module depends on EDIpack2 (which ultimately depends on SciFortran) and Triqs. Assuming the two software are correctly installed in the OS, the EDIpack2Triqs interface is installed as follows:

```
git clone https://github.com/krivenko/edipack2triqs
cd edipack2triqs
pip install .
```

4.4. W2Dynamics interface

`sSecInteropW2DYN`

Giorgio, Alexander,?

4.5. EDIjl, the Julia API

`sSecInteropEDIjl`

Lorenzo?

5. Examples

`SecExamples`

In this section we illustrate the functioning of the EDIpack2 library and their interfaces through specific examples, including discussion of relevant code snippets. Specifically, we show how the quantum impurity solver in EDIpack2 can be used to address problems of different nature within the framework of DMFT.

5.1. Bethe lattice DMFT (Fortran API)

The description of the Mott or metal to insulator transition (MIT) within the Bethe lattice, i.e. Cayley tree with connection $z \to \infty$ is conventionally considered the test bed of any DMFT application. Here we discuss a guided implementation based on the Fortran API of EDIpack2 of the DMFT solution of the Bethe lattice at $T = 0$.

We consider a Fermi-Hubbard model:

$$H = -t \sum_{\langle ij \rangle \sigma} c^+_{i\sigma} c_{j\sigma} + U \sum_i n_{i\uparrow} n_{i\downarrow}$$

defined on a Bethe lattice with density of states $\rho(x) = \frac{1}{2D}\sqrt{D^2 - x^2}$, where $D = 2t$ is the half-bandwidth.

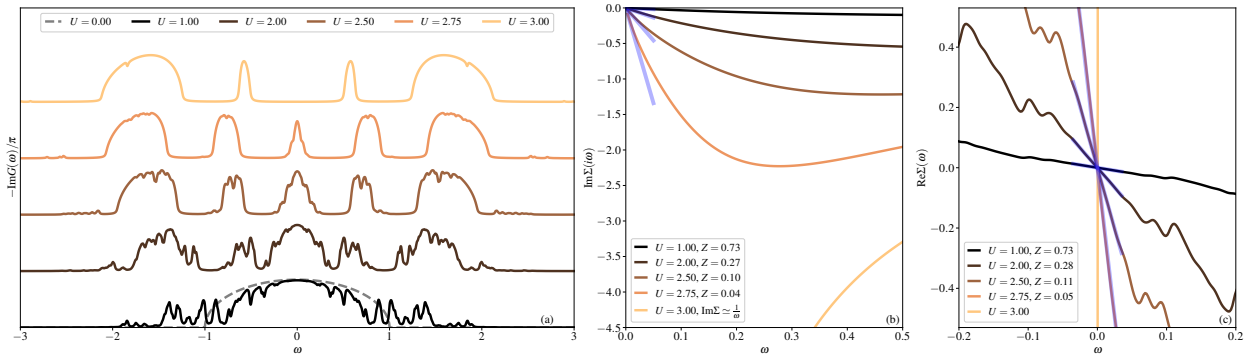

fig1

Figure 1: **The Mott transition.**

5.2. Attractive Hubbard model (Python API)

5.3. Multi-orbital Hubbard (Triqs)

6. Conclusions

Acknowledgements

7. Appendix A: Monicelli interface

References

[1] A. Georges, L. de' Medici, J. Mravlje, Strong Correlations from Hund's Coupling., Annu. Rev. Condens. Matter Phys. 45 (2013) 137–178.

[2] C. Lanczos, An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, J. Res. Natl. Bur. Stand. B 45 (1950) 255–282. doi:10.6028/jres.045.026.

[3] H. Lin, J. Gubernatis, Exact diagonalization methods for quantum systems, Computers in Physics 7 (4) (1993) 400–407.
URL https://doi.org/10.1063/1.4823192

[4] R. Lehoucq, D. Sorensen, C. Yang, ARPACK Users' Guide: Solution of Large-scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods, Software, Environments, Tools, Society for Industrial and Applied Mathematics, 1998.
URL https://books.google.it/books?id=iMUea23N_CQC

[5] K. J. "Maschhoff, D. C. Sorensen, P_arpack: An efficient portable large scale eigenvalue package for distributed memory parallel architectures, in: J. "Waśniewski, J. Dongarra, K. Madsen, D. Olesen (Eds.), Applied Parallel Computing Industrial Computation and Optimization, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 478–486.

[6] A. Georges, G. Kotliar, W. Krauth, M. J. Rozenberg, Dynamical mean-field theory of strongly correlated fermion systems and the limit of infinite dimensions, Rev. Mod. Phys. 68 (1996) 13–125. doi:10.1103/RevModPhys.68.13.
URL https://link.aps.org/doi/10.1103/RevModPhys.68.13

[7] D. J. García, K. Hallberg, M. J. Rozenberg, Dynamical mean field theory with the density matrix renormalization group, Phys. Rev. Lett. 93 (2004) 246403. doi:10.1103/PhysRevLett.93.246403.
URL https://link.aps.org/doi/10.1103/PhysRevLett.93.246403

[8] C. Taranto, G. Sangiovanni, K. Held, M. Capone, A. Georges, A. Toschi, Signature of antiferromagnetic long-range order in the optical spectrum of strongly correlated electron systems, Phys. Rev. B 85 (2012) 085124. doi:10.1103/PhysRevB.85.085124.
URL https://link.aps.org/doi/10.1103/PhysRevB.85.085124

[9] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, Numerical Recipes in FORTRAN 77: The Art of Scientific Computing, 2nd Edition, Cambridge University Press, 1992.
URL http://www.worldcat.org/isbn/052143064X