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. Scazzola[e], F. Petocchi[f], G. Mazza[g], L. de Medici[h], M. Capone[d], 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

*Corresponding author:giuli@sissa.it
**Corresponding author:bellomia@sissa.it

1. Introduction and Motivation

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. Structure and derived software

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 interface, which enables development of additional API or inter-operability with external libraries.

- **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.4.

- **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 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.
- **EDIpy2.** This is a simple Python module which provides Python API to the EDIpack2 Fortran library. This interface is built around the Python support to C-types, which allows to import the dynamic C-binding library generated upon building EDIpack2 . The module contains a specific class, whose methods mirrors through duck-typing all the available procedures of EDIpack2 as well as it gives access to relevant shared control variables.
- **EDIpack2Triqs.** This is a thin interface layer from EDIpack2 to Triqs, built around the Python API of the EDIpack2 library. The exact diagonalization solver is encapsulated in a dedicated class, containing the necessary methods to initialize and run a single instance of the solver. The interface also includes a specific class encompassing the effective discretized bath structure as well as their optimization methods.

3. Installation

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.

3.1. Dependencies

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.

3.2. Build and Install

3.2.1. 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.

3.2.2. 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.

3.3. OS Loading

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.

3.4. Python API

3.4.1. Build from source

EDIpy2, i.e. the Python API of EDIpack2, is available as a stand-alone module which depends on both EDIpack2 and SciFortran. The 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.

3.4.2. Anaconda

As for EDIpack2, also the Python API in EDIpy2 are available through Anaconda packaging. In this case the resolution of the dependencies is taken care from Conda itself:

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

3.5. TRIQS interface

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. Implementation

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

4.1. The quantum impurity problem

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:

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

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:

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

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:

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

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, to cut-off the unbounded dimensions of the local phonons Hilbert space.

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`$= N_\sigma$, `Norb`$= N_\alpha$ 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. 4.9.

4.2. The Fock basis states

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, $\mathcal{H}_{ph} = \{|0\rangle, |1\rangle, \ldots, |N_{ph}\rangle\}$ 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

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

where $p = 1, \ldots, N_{ph}$ 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 $I = 0, \ldots 2^{2N_s} - 1$ 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:

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

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.

4.3. Conserved quantum numbers

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:

- **normal:** $\vec{Q} = [N, S_z] \equiv [N_\uparrow, N_\downarrow]$
- **superc:** $\vec{Q} = S_z$
- **nonsu2:** $\vec{Q} = N$

The **normal** case consider 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. According to the previously introduced symmetries we have:

- **normal:** $D_\mathcal{S} = \binom{N_s}{N_\uparrow}\binom{N_s}{N_\downarrow}$
- **superc:** $D_\mathcal{S} = \sum_m 2^{N_s - S_z - 2m}\binom{N_s}{N_s - S_z - 2m}\binom{S_z+2m}{m}$
- **nonsu2:** $D_\mathcal{S} = \binom{2N_s}{N}$

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 symmetry sector reduction of the Fock space induces a block digonal form to the Hamiltonian matrix. Each block, labelled by a value of 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) = lup
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`.

4.4. 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. Text to be revised

4.4.1. Sparse matrix

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.

4.4.2. Eigenspace

This class, contained in `ED_EIGENSPACE` implements a dedicated storage for the eigenvalues and eigenvectors of the quantum impurity Hamiltonian The 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.

4.4.3. GFmatrix

One of the main goal of the code is to evaluate dynamical correlations functions $\langle \mathcal{T}_{\pm}[A(t)A^{+}]\rangle$. 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 $\frac{1}{Z}\sum_n e^{-\beta E_n}\sum_{m=1}^{N}\frac{|w_{mn}|^2}{z-dE_{mn}} \equiv$ where w_{mn} is a weight set by the projection of the $m^{\rm th}$ eigenstate onto the $n^{\rm th}$ 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 efficiently storing all the *weights* w_{mn} and *poles* dE_{mn} contributing to a specific dynamical correlation functions. Specifically, the `gfmatrix` object implements a dynamical multi-layer data structure gatering weights and poles from any initial state and for any combination of operators contributing to the dynamical correlation function.

The use of this class enables the istantan

4.5. Lanczos based Diagonalization

4.6. Dynamical correlation functions

4.7. Observables

4.8. Reduced impurity density matrix

4.9. Bath parametrization

4.10. Bath Optimization

4.11. Input/Output

5. Inequivalent impurities

6. C-bindings

7. Python API

8. Triqs interface

9. Usage

9.1. Bethe lattice DMFT (Fortran API)

9.2. Attractive Hubbard model (Python API)

9.3. Multi-orbital Hubbard (Triqs)

10. Conclusions

Acknowledgements

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.