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

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: .

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

2.1. EDIpack2

2.2. EDIpack2ineq

2.3. EDIpack2 C-bindings

2.4. EDIpack2Triqs

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: `libedipack.a` and `libedipack_cbinding.so`. The former, alongside the generated Fortan modules files, wraps the EDIpack2 software. The latter enables interoperability through bindings to the C language.

3.1. Dependencies

Although EDIpack2 is as much self-contained as possible its development hinges 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. Usage

4.1. Bethe lattice DMFT (Fortran API)

4.2. Attractive Hubbard model (Python API)

4.3. Multi-orbital Hubbard (Triqs)

5. Implementation

5.1. The quantum impurity problem

5.2. The basis states

5.3. Conserved quantum numbers

5.4. Krylov based Diagonalization

5.5. Dynamical correlation functions, observables and reduced impurity density matrix

5.6. Bath parametrization

5.7. Bath χ^2 fit

5.8. Input/Output

6. Inequivalent impurities

7. C-bindings

8. Python API

9. EDIpack2triqs: a thin interface to Triqs

10. Conclusions

Acknowledgements

References