Computer Physics Communications 243 (2019) 151–165

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

# EDRIXS: An open source toolkit for simulating spectra of resonant inelastic x-ray scattering☆

Y.L. Wang [a,*], G. Fabbris [a], M.P.M. Dean [a], G. Kotliar [a,b]

[a] Department of Condensed Matter Physics and Materials Science, Brookhaven National Laboratory, Upton, NY 11973, USA
[b] Department of Physics and Astronomy, Rutgers University, Piscataway, NJ 08856, USA

ARTICLE INFO

*Article history:*
Received 27 December 2018
Received in revised form 16 April 2019
Accepted 16 April 2019
Available online 21 May 2019

*Keywords:*
Resonant inelastic x-ray scattering
Cross-section
Exact diagonalization

ABSTRACT

Resonant inelastic x-ray scattering (RIXS) has become a very powerful experimental technique to probe a broad range of intrinsic elementary excitations, for example, from low energy phonons and (bi-)magnons to high energy *d-d*, charge-transfer and plasmon excitations in strongly correlated electronic systems. Due to the complexity of the RIXS cross-section and strong core-hole effects, theoretical simulation of the experimental RIXS spectra is still a difficult task which hampers the understanding of RIXS spectra and the development of the RIXS technique. In this paper, we present an open source toolkit (dubbed EDRIXS) to facilitate the simulations of RIXS spectra of strongly correlated materials based on exact diagonalization (ED) of certain model Hamiltonians. The model Hamiltonian can be from a single atom, small cluster or Anderson impurity model, with model parameters from density functional theory plus Wannier90 or dynamical mean-field theory calculations. The spectra of x-ray absorption spectroscopy (XAS) and RIXS are then calculated using Krylov subspace techniques. This toolkit contains highly efficient ED, XAS and RIXS solvers written in modern Fortran 90 language and a convenient Python library used to prepare inputs and set up calculations. We first give a short introduction to RIXS spectroscopy, and then we discuss the implementation details of this toolkit. Finally, we show three examples to demonstrate its usage.

**Program summary**
*Program title:* EDRIXS
*Program Files doi:* http://dx.doi.org/10.17632/wvd5x3mtg4.1
*Licensing provisions:* GNU General Public Licence 3.0
*Programming language:* Fortran 90 and Python3
*External routines/libraries used:* BLAS, LAPACK, Parallel ARPACK, numpy, scipy, sympy, matplotlib, sphinx, numpydoc
*Nature of problem:* Simulating the spectra of resonant inelastic x-ray scattering for strongly correlated electronic systems.
*Solution method:* Exact diagonalization of model Hamiltonians and Krylov subspace methods.


## 1. Introduction

Directly measuring the elementary excitations in materials through experimental spectroscopy is a fundamental task in condensed matter physics. For example, angle-resolved photoemission spectroscopy (ARPES) [1] is a very powerful technique to measure the single particle spectrum function $A(\vec{k}, \omega)$ (bands) in both weakly and strongly correlated materials. In recent years, resonant inelastic x-ray scattering (RIXS) has also emerged as a key probe of the elementary excitations in materials, especially for strongly correlated systems [2]. RIXS is a *photon-in photon-out* spectroscopy. The energy of the incident photon is tuned at a specific resonant edge to excite a deep core electron to the valence shell and create a core-hole. After a very short time, the core-hole is refilled by a valence electron and photon is emitted. By measuring the change in energy, momentum and polarization of the scattered photon, one can obtain detailed

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655)..

* Corresponding author.
*E-mail address:* yilinwang@bnl.gov (Y.L. Wang).

https://doi.org/10.1016/j.cpc.2019.04.018
0010-4655/

information on the nature of the underlying excitations. Compared to other spectroscopies, RIXS has various unique features and advantages [2]. First, it can measure a broad range of elementary excitations, for example, from low energy phonons and (bi-)magnons to high energy local *d*-*d*, charge transfer (CT) and plasmon excitations. Second, since x-rays carry appreciable momentum, the dispersion of low-energy excitations in solids can be probed. Third, there are fewer restrictions to sample surface quality or volume compared to probes such as ARPES and neutron scattering, which enables, for instance, studies of thin films and very small single crystals. Finally, it is element and orbital specific, bulk sensitive and polarization dependent. In recent years, great progress in the technical development of RIXS has occurred due to high-brilliance synchrotron light sources and advanced photon detection instrumentation, leading to its wide use to study elementary excitations in strongly correlated materials [2–6].

Given the RIXS sensitivity to a wide range of elementary excitations, theoretical simulation of the RIXS cross-section is often very relevant in the interpretation of the experimental RIXS spectra and its underlying physics. The RIXS cross-section is described by the Kramers–Heisenberg formula, which is a result of second-order perturbation theory. The RIXS cross-section can be interpreted as a four-particle correlation function involving both valence and core electrons [7,8] and can be in principle determined by some first-principles numerical methods. Nowadays, density functional theory plus dynamical mean-field theory (DFT+DMFT) [9,10] with continuous-time quantum Monte-Carlo (CTQMC) [11] as the impurity solver is the most powerful and accurate method for first-principle calculations of physical properties, such as the single particle spectrum function, of strongly correlated materials, so it is a desirable method to compute RIXS spectra. Unfortunately, it is still a very challenging mission to determine the RIXS cross-section by this method with the current available numerical techniques due to the complexity of this correlation function and the strong unknown core-hole potential. The difficulty in performing theoretical RIXS simulations hampers the progress of the understanding of RIXS spectra and the development of the RIXS technique. Nowadays, most of RIXS simulations are based on the exact diagonalization (ED) of Hamiltonians of small size models, that capture some sorts of the properties of the real materials, such as single atom, small cluster and Anderson impurity model (AIM) [12]. Currently, the most widely used RIXS simulation codes based on ED algorithms include CTM4RIXS [13] and Quanty [14]. The former is based on an old atomic code called Cowan's code [15] with very limited functionalities to perform XAS and RIXS calculations. The latter is based on a lightweight programming language called Lua [16] and has much richer functionalities. However, the core of the Quanty code is not open source, so it cannot be freely modified by users to implement functionalities that are not available. Many RIXS simulations based on ED [17–37] or DMRG [38] have been also reported. In Refs. [24,27,29], the ED code is based on the parallel ARPACK library. In Refs. [35], a powerful ED solver based on quantum chemistry algorithm is used to diagonalize the AIM from a converged DFT+DMFT calculations with many bath sites and continuous charge excitations have been found in high-valence transition-metal oxides in their simulated RIXS spectra. However, currently these codes are not open source.

In this work, we present the EDRIXS open source toolkit that is aimed at facilitating RIXS simulations based on ED algorithms. EDRIXS is designed as a post-processing tool for the open source COMSCOPE project [39,40]. It implements parallel ED solvers with very high efficiency based on the standard Lanczos algorithms and parallel version of ARPACK library. The XAS and RIXS spectra are then calculated parallelly using Krylov subspace algorithms. The core components of this toolkit are implemented using the modern Fortran 90 language and the parallelism is achieved by MPI. The code has a layered structure, thus new functionalities can be easily implemented by users who want to perform other types of x-ray scattering simulations. An application programming interface (API) is implemented by the popular Python3 language to prepare inputs and set up calculations. A pure Python ED solver is also implemented in the API for small size problem. The rest of this paper is organized as follows: In Section 2, we introduce the basic theory and methods of RIXS simulations. In Section 3, we explain the implementation of the code in detail. In Section 4, we show how to install and use this code. In Section 5, we show three examples to demonstrate the usage of this code. In Section 6, we discuss the plans of future development of EDRIXS.

## 2. Basic theory and methods of RIXS simulation

### 2.1. Geometry of RIXS experiment

A typical geometry of RIXS experiment is illustrated in Fig. 1. $\vec{k}_i$ ($\vec{\epsilon}_i$) and $\vec{k}_f$ ($\vec{\epsilon}_f$) are the wave (polarization) vectors of the incident and scattered x-ray, respectively. $\omega_{\text{in}}$ and $\omega_{\text{out}}$ are the energy of incident and scattered x-ray, respectively. $\alpha$ and $\beta$ are the angles between the polarization vectors and the scattering plane. We call it $\pi$-polarization when the polarization vector is parallel to the scattering plane ($\alpha, \beta = 0$), and $\sigma$-polarization when the polarization vector is perpendicular to the scattering plane ($\alpha, \beta = \pi/2$). $\theta_{\text{in}}$ and $\theta_{\text{out}}$ are the incident and scattered angles, respectively. $\phi$ is the azimuthal angle defined with respect to *x*-axis.

### 2.2. XAS and RIXS cross-section

The XAS cross-section is described by

$$I_{\text{XAS}}(\omega_{\text{in}}, \vec{\epsilon}_i) = \sum_i \frac{1}{Z} e^{-\frac{E_i}{K_B T}} \sum_n \left|\langle n|\hat{\mathcal{D}}_i|i\rangle\right|^2 \frac{\Gamma_c/\pi}{(\omega_{\text{in}} - E_n + E_i)^2 + \Gamma_c^2}, \tag{1}$$

and the RIXS cross-section is described by the Kramers–Heisenberg formula [2],

$$I_{\text{RIXS}}(\omega_{\text{in}}, \omega, \vec{k}_i, \vec{k}_f, \vec{\epsilon}_i, \vec{\epsilon}_f) = \sum_i \frac{1}{Z} e^{-\frac{E_i}{K_B T}} \sum_f \left|\left\langle f \left| \hat{\mathcal{D}}_f^\dagger \frac{1}{\omega_{\text{in}} - \hat{H}_n + E_i + i\Gamma_c} \hat{\mathcal{D}}_i \right| i \right\rangle\right|^2 \times \frac{\Gamma/\pi}{(\omega - E_f + E_i)^2 + \Gamma^2}. \tag{2}$$

where $|i\rangle$ are the ground states of the Hamiltonian $\hat{H}_i$ describing the valence electrons. $T$ is temperature and $K_B$ is the Boltzmann factor. $Z = \sum_i e^{-\frac{E_i}{K_B T}}$ is the partition function. $|f\rangle$ are the excited eigenstates of $\hat{H}_i$. $|n\rangle$ are the eigenstates of intermediate Hamiltonian $\hat{H}_n$

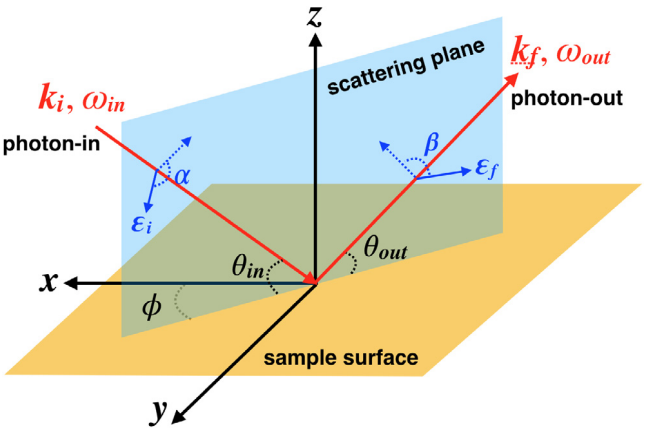


**Fig. 1.** The illustration of the geometry of RIXS experiment.

which includes a deep core-hole and one more electron in the valence shell after the x-ray absorption process. $\omega_{\text{in}}$ is the energy of the incident x-ray which is tuned to be at a specific resonant edge. $\omega$ is the x-ray energy loss, i.e. the energy difference between the incident and scattered x-ray. $\Gamma_c$ and $\Gamma$ are the lifetime broadening of the intermediate (with a core hole) and final states, respectively.

$\hat{\mathcal{D}}_i$ and $\hat{\mathcal{D}}_f^\dagger$ are the transition operators for x-ray absorption and emission process, respectively. They can be generally written as

$$\hat{\mathcal{D}}_i = \sum_a P_a^i \hat{T}_a^i, \tag{3}$$

$$\hat{\mathcal{D}}_f^\dagger = \sum_a P_a^{f\star} \hat{T}_a^{f\dagger}. \tag{4}$$

where $P_a^i$ and $P_a^f$ involve the polarization and wave vector of x-ray. For dipolar transition, they are: $\epsilon_x^i, \epsilon_y^i, \epsilon_z^i$ and $\epsilon_x^f, \epsilon_y^f, \epsilon_z^f$. $\hat{T}_a^i$ and $\hat{T}_a^f$ are components of transition operators. For dipolar transition, they are:

$$\hat{T}_x^i = \sum_R e^{i\vec{k}_i\cdot\vec{R}}\hat{x}_R, \quad \hat{T}_y^i = \sum_R e^{i\vec{k}_i\cdot\vec{R}}\hat{y}_R, \quad \hat{T}_z^i = \sum_R e^{i\vec{k}_i\cdot\vec{R}}\hat{z}_R, \tag{5}$$

$$\hat{T}_x^f = \sum_R e^{i\vec{k}_f\cdot\vec{R}}\hat{x}_R, \quad \hat{T}_y^f = \sum_R e^{i\vec{k}_f\cdot\vec{R}}\hat{y}_R, \quad \hat{T}_z^f = \sum_R e^{i\vec{k}_f\cdot\vec{R}}\hat{z}_R, \tag{6}$$

where $\vec{R}$ is site-index, and $\hat{x}_R$, $\hat{y}_R$ and $\hat{z}_R$ are position operators of electrons bound to site-$R$. These are measured with respect to the center of site-$R$.

### 2.3. Model hamiltonian

$\hat{H}_i$ is a model Hamiltonian for valence electrons from single atom, cluster or Anderson impurity models. It can be generally written as

$$\hat{H}_i = \sum_{\alpha,\beta} t_{\alpha,\beta}\hat{f}_\alpha^\dagger\hat{f}_\beta + \sum_{\alpha,\beta,\gamma,\delta} U_{\alpha,\beta,\gamma,\delta}\hat{f}_\alpha^\dagger\hat{f}_\beta^\dagger\hat{f}_\gamma\hat{f}_\delta, \tag{7}$$

where $t_{\alpha,\beta}$ usually contain crystal field (CF) splitting terms, spin–orbit coupling (SOC) terms, hopping terms between different sites. The CF and hopping terms can be obtained from a first-principle DFT+Wannier90 calculations. SOC can be obtained from DFT calculations or single atomic calculation, for example, from Cowan's atomic code [15]. $U_{\alpha,\beta,\gamma,\delta}$ are the rank-4 tensors of the on-site Coulomb interaction and are parameterized by Slater integrals, which can be obtained from an atomic calculations.

$\hat{H}_n$ is the intermediate Hamiltonian with a core-hole, which can be written as

$$\hat{H}_n = \hat{H}_i + \hat{V}_{\text{core-hole}} + \hat{H}_{\text{core}}, \tag{8}$$

where the core-hole potential $\hat{V}_{\text{core-hole}}$ is simulated at atomic level by adding Coulomb interaction between valence electrons and core-hole. $\hat{H}_{\text{core}}$ is the Hamiltonian of the core electrons.

To simulate RIXS spectra based on DFT+DMFT calculations, we first perform a DFT+DMFT calculation to get converged hybridization function $\Delta_\alpha(i\omega_n)$ and then construct an Anderson impurity model $\hat{H}_i$ from it, which can be written as

$$\hat{H}_i = \hat{H}_{\text{imp}} + \sum_{\alpha,l} E_{\alpha,l}\hat{c}_{\alpha,l}^\dagger\hat{c}_{\alpha,l} + \sum_{\alpha,l} V_\alpha^l\hat{f}_\alpha^\dagger\hat{c}_{\alpha,l} + h.c., \tag{9}$$

where $\hat{H}_{\text{imp}}$ is the Hamiltonian for the impurity site, which is taken from the atomic problem in DFT+DMFT calculation. $E_{\alpha,l}$ is the energy level of the $l$th bath site with orbital $\alpha$, $V_\alpha^l$ are the hybridization strength between localized impurity electrons ($\hat{f}^\alpha$) and conducted bath

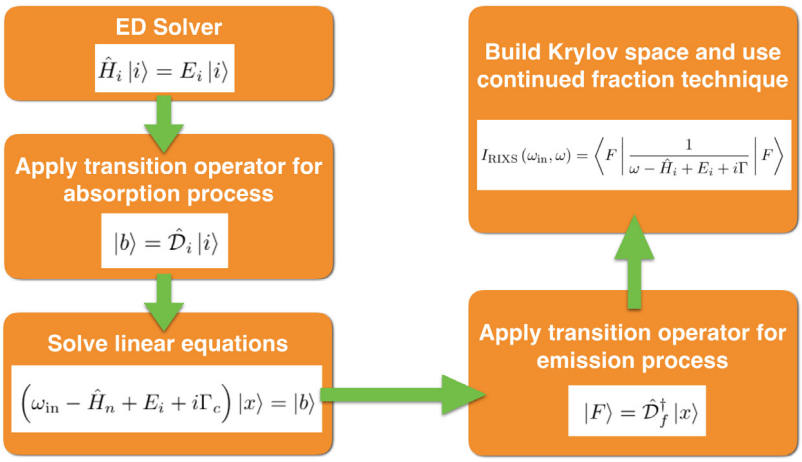


**Fig. 2.** The flow diagram of algorithm to calculate RIXS cross-section.

electrons ($\hat{c}_{\alpha,l}$). $E_{\alpha,l}$ and $V_\alpha^l$ are obtained by fitting the hybridization function,

$$\Delta_\alpha(i\omega_n) = \sum_{l=1}^{N} \frac{\left|V_\alpha^l\right|^2}{i\omega_n - E_{\alpha,l}}. \tag{10}$$

The core-hole potential is only added to the impurity site in $\hat{H}_n$.

### 2.4. RIXS simulation algorithms

We use similar RIXS simulation algorithms as that used in Ref. [24,35]. The flow diagram of the algorithm to calculate the RIXS cross-section is illustrated in Fig. 2 and briefly described as follows:

- Use an ED solver to diagonalize $\hat{H}_i$ to get the ground states or a few low-energy excited states $|i\rangle$ ($E_i$). Both standard Lanczos algorithm [41] without re-orthogonalization and implicitly restarted Arnoldi algorithm [42,43] are used in ED solvers.
- Apply the transition operator for the x-ray absorption process on $|i\rangle$ to get $|b\rangle = \hat{\mathcal{D}}_i |i\rangle$.
- Solve the following large-scale sparse symmetric linear equations,

$$\left(\omega_{\rm in} - \hat{H}_n + E_i + i\Gamma_c\right)|x\rangle = |b\rangle\,, \tag{11}$$

  by minimum residual (MINRES) method [44].
- Apply the transition operator for the x-ray emission process on $|x\rangle$ to get $|F\rangle = \hat{\mathcal{D}}_f^\dagger |x\rangle$.
- Build Krylov space of $\hat{H}_i$ and use continued fraction technique to get the RIXS spectra,

$$I_{\rm RIXS}(\omega_{\rm in},\omega) = \left\langle F \left| \frac{1}{\omega - \hat{H}_i + E_i + i\Gamma} \right| F \right\rangle. \tag{12}$$

## 3. Implementations and optimizations

In this section, we describe the details of implementation and optimization of EDRIXS.

### 3.1. Development platform

The key components of the EDRIXS code are developed with the modern Fortran 90 language. Intel's *ifort* and GNU's *gfortran* compilers have been tested. The code is parallelized by MPI. An interface that is used for pre- and post-processing is written in the Python3 language. Git is used as the version control system.

### 3.2. Fock basis

We implement EDRIXS code based on the second-quantization language. We first define a single particle basis $\alpha$, and then define Fock basis $|I\rangle$ with respect to it. In the code, all the orbitals of valence electrons are gathered together and put in front of all the orbitals of core electrons (i.e. indices of valence orbitals are always smaller than those of core orbitals). A Fock state can be represented by a binary number with digital 1 or 0, where 1 means that the orbital is occupied and 0 means that the orbital is empty. For example,

$$\overbrace{110100}^{1-6}\ \overbrace{111011}^{7-12} \tag{13}$$

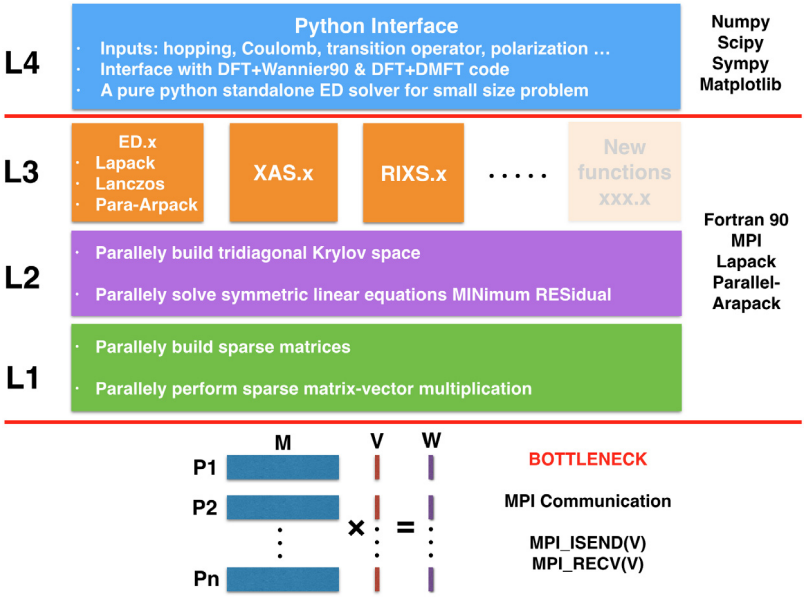


**Fig. 3.** The structure of EDRIXS code.

where the red parts with orbital indices from 1 to 6 are for valence electrons and the blue parts with orbital indices from 7 to 12 are for core electrons. In this example, valence orbitals 1, 2 and 4 are occupied, and one core orbital 10 is empty. The above binary format is used for the ED solver implemented in Python API. For *ed.x*, *xas.x* and *rixs.x*, we need to input the Fock basis with their corresponding decimal formats, but only for the parts of valence orbitals. The core orbitals will be explicitly considered within these three programs. For the above example (Eq. (13)), the corresponding decimal number is computed as follows:

$$1 \times 2^{1-1} + 1 \times 2^{2-1} + 0 \times 2^{3-1} + 1 \times 2^{4-1} + 0 \times 2^{5-1} + 0 \times 2^{6-1} = 11. \tag{14}$$

### 3.3. Format of sparse matrices

In second-quantization form, a general operator can be written as

$$\hat{O} = \sum_{\alpha,\beta} t_{\alpha,\beta} \hat{f}^{\dagger}_{\alpha} \hat{f}_{\beta} + \sum_{\alpha,\beta,\gamma,\delta} U_{\alpha,\beta,\gamma,\delta} \hat{f}^{\dagger}_{\alpha} \hat{f}^{\dagger}_{\beta} \hat{f}_{\gamma} \hat{f}_{\delta}, \tag{15}$$

where we call $t_{\alpha,\beta}$ hopping terms and $U_{\alpha,\beta,\gamma,\delta}$ rank-4 interaction terms. The matrix elements of $\hat{O}$ in the Fock basis $|I\rangle$ are $\left\langle I' \middle| \hat{O} \middle| I \right\rangle$. These matrices are large-scale sparse matrices and the number of nonzero elements is proportional to its dimension $n$, so we store them in compressed sparse row (CSR) format to save memory. To speed up the building of operators, only nonzeros of $t_{\alpha,\beta}$ and $U_{\alpha,\beta,\gamma,\delta}$ are input.

### 3.4. Structure of the code

The structure of the EDRIXS code can be divided into four layers, which is illustrated in Fig. 3. The key components of EDRIXS code involve two basic operations: building matrices form of operators in Fock basis and performing sparse matrix–vector multiplication (MVM), which are the most time-consuming parts, so we implement subroutines to parallelly do these tasks in the first layer. The rows of a sparse matrix $M$ and elements of a vector $V$ are equally distributed among all the processors. Each processor only needs to build parts of the rows of the operator and do parts of the MVM, and all the processors can work parallelly. The only MPI communication needed for MVM is to send and receive the vector $V$, which is a bottleneck of the code. The speed of this MPI communication heavily depends on the networking of the cluster. Based on the first layer, we implement subroutines to parallelly build tridiagonal Krylov space and solve symmetric linear equations using MINRES methods in the second layer. The key operations of these two subroutines are MVM.

Above the first and second layers, we implement the main functionalities: *ed.x*, *xas.x* and *rixs.x* to do RIXS simulations in the third layer. Three types of ED solvers are implemented. The first one is to use the LAPACK's subroutine ZHEEV to diagonalize a Hamiltonian to get all the eigenstates, which is useful for small size ($n < 10,000$) problem. The second one is to use standard Lanczos algorithm without re-orthogonalization to get only one ground state, which is not very accurate due to the orthogonality problem, especially for degenerate ground states. However, it runs very fast and is suitable for roughly searching the ground state of a Hamiltonian in many different subspaces. The third one is to use the parallel version of the ARPACK library to get few lowest eigenstates, which is very accurate because it uses implicitly restarted Arnoldi algorithm to deal with the orthogonalization issue, but its efficiency is lower than the second one. We will use this solver in real RIXS simulations. Based on the first and second layers, we can easily extend new functionalities to do more calculations, for example, calculating the expected value of an operator in the ground state and other correlation functions.

**Table 1**
Requirement of tools and external libraries for building EDRIXS code.

| Tools and Libraries | Recommendation |
|---|---|
| Fortran compiler | Intel's ifort or GNU's gfortran |
| MPI | OpenMPI or MPICH |
| BLAS/LAPACK | MKL or OpenBLAS [47] |
| Parallel ARPACK | arpack-ng [48], v3.6.2 or higher |
| Python3 | v3.6 or higher |
| Numpy | v1.15 or higher |
| Scipy | v1.1 or higher |
| Sympy | v1.3 or higher |
| Matplotlib | v3.0 or higher |
| Sphinx | v1.7 or higher |
| Numpydoc | v0.8 or higher |

In the fourth layer, a Python interface is implemented to provide some application programming interfaces (APIs) for preparing inputs for *ed.x*, *xas.x* and *rixs.x* and setting up calculations. Besides, a pure python ED solver based on second-quantization is also implemented for small size problem, so the RIXS simulations can be done easily with pure Python code. Currently, we have not implemented the interface with the Cowan's code for calculating Slater integrals. One should manually run Cowan's code and read Slater integrals. We refer users to its official webpage [45] for more information.

## 4. Installation and usage

In this section, we will show how to install and use EDRIXS on Linux or OSX platforms. We encourage the readers to check https://github.com/NSLS-II/edrixs for the most up to date information.

### 4.1. Get EDRIXS

EDRIXS is an open source code and is released under the GNU General Public Licence 3.0 (GPL). The readers who are interested in it can write a letter to the authors to request a copy of the newest version of EDRIXS source code, or they can download it directly from the public code repository:

```
$ git clone https://github.com/NSLS-II/edrixs.git
```

where $ is the command line prompt. One can use the command

```
$ git pull
```

to get the latest updates of the code.

### 4.2. Build EDRIXS

To build the source code and documentation of EDRIXS, some tools and external libraries are required, which are listed in Table 1. Most of the libraries can be easily installed using *Anaconda* [46]. There may be problems when using *gfortran* with *MKL*, so we strongly recommend *gfortran+OpenBLAS*. Note that the *arpack-ng* library should be compiled with the same Fortran compiler and *BLAS/LAPACK* libraries.

We assume that one downloads the source code into the directory *EDRIXS_DIR*. One then go to the *src* directory and edit the *make.sys* file to configure the compiling settings.

```
$ cd src
$ cp make.sys.ifort make.sys  (or cp make.sys.gfortran make.sys)
$ vim make.sys
```

where, in the *make.sys* file, one needs to set F90 as

```
F90 = mpif90
```

and set LIBS, for example,

```
LIBS = -L${MKLROOT}/lib/intel64 -lmkl_core -lmkl_sequential -lmkl_rt \\
    -L${ARPACK_DIR}/lib/ -lparpack -larpack
```

where we assume that the *ARPACK* library is installed in *ARPACK_DIR*. Then, we type

```
$ make
$ make install
```

to compile and install the executable files *ed.x*, *xas.x* and *rixs.x* into *bin* directory. Then, one should add the following two lines in *.bashrc* or *.bash_profile* file,

```
export PATH=${EDRIXS_DIR}/bin:$PATH
```

To install the Python APIs of EDRIXS, one should just type

```
$ python setup.py install
```

**Table 2**
Input and output files for *ed.x*.

| | File Name | Description |
|---|---|---|
| Inputs | config.in | set up control parameters |
| | fock_i.in | Fock basis $\|I\rangle$ for $\hat{H}_i$ |
| | hopping_i.in | $t_{\alpha,\beta}$ terms in $\hat{H}_i$ |
| | coulomb_i.in | $U_{\alpha,\beta,\gamma,\delta}$ terms in $\hat{H}_i$ |
| Outputs | standard outputs | log file |
| | eigvals.dat | eigenvalues $E_i$ |
| | eigvec.*i* | the *i*th ground state $\|\Gamma_i\rangle$ |
| | denmat.dat | density matrix: $\left\langle \Gamma_i \middle\| \hat{f}_\alpha^\dagger \hat{f}_\beta \middle\| \Gamma_i \right\rangle$ |

**Table 3**
Input and output files for *xas.x*.

| | File Name | Description |
|---|---|---|
| Inputs | config.in | set up control parameters |
| | fock_i.in | Fock basis $\|I\rangle$ for $\hat{H}_i$ |
| | fock_n.in | Fock basis $\|I\rangle$ for $\hat{H}_n$ |
| | hopping_n.in | $t_{\alpha,\beta}$ terms in $\hat{H}_n$ |
| | coulomb_n.in | $U_{\alpha,\beta,\gamma,\delta}$ terms in $\hat{H}_n$ |
| | transop_xas.in | transition operator $\hat{\mathcal{D}}_i$ |
| | eigvec.*i* | the *i*th ground state $\|\Gamma_i\rangle$ of $\hat{H}_i$ |
| Outputs | standard outputs | log file |
| | xas_poles.*i* | XAS data for the *i*th ground state |

To compile the documentation of the Python APIs, one should

```
$ cd docs
$ mkdir build
$ sphinx-build -b html source build
$ make html
```

and open the file

```
${EDRIXS_DIR}/docs/build/index.html
```

in a browser to read the documentation of the Python APIs.

### 4.3. Use EDRIXS

The typical procedures of the usage of EDRIXS are as follows:

- Use Python APIs to set up the parameters of Hamiltonian, transition operators, Fock basis.

```
import edrixs
edrixs.some_functions()
```

- Run *ed.x*, *xas.x* and *rixs.x* in order. These three Fortran executable files can be launched manually, for example,

```
mpirun -np 16 ed.x > log.dat
```

  or through the Python *subprocess* call, for example,

```
import subprocess
subprocess.check_call([‘‘mpirun", ‘‘-np", ‘‘16", ‘‘ed.x"])
```

- Use Python APIs to post-process the results and plot XAS or RIXS spectra.

The required input and output files for *ed.x*, *xas.x* and *rixs.x* are listed in Tables 2–4, respectively. More details about the input and output files are described in file *user_guide.pdf* in *docs* directory.

## 5. Examples

In this section, we show three examples to demonstrate the usage of EDRIXS. The testing platform is a Macbook laptop (CPU: 2.9 GHz Intel Core i5, Memory: 8 GB 1867 MHz DDR3). The codes are compiled by Intel's ifort compiler with MKL and arpack-ng. The approximate total running times of these three examples are summaried in Table 5.

**Table 4**
Input and output files for *rixs.x*.

| | File Name | Description |
|---|---|---|
| Inputs | config.in | set up control parameters |
| | fock_i.in | Fock basis $\|I\rangle$ for $\hat{H}_i$ |
| | fock_n.in | Fock basis $\|I\rangle$ for $\hat{H}_n$ |
| | fock_f.in | Fock basis $\|I\rangle$ for $\hat{H}_i$ |
| | hopping_i.in | $t_{\alpha,\beta}$ terms in $\hat{H}_i$ |
| | hopping_n.in | $t_{\alpha,\beta}$ terms in $\hat{H}_n$ |
| | coulomb_i.in | $U_{\alpha,\beta,\gamma,\delta}$ terms in $\hat{H}_i$ |
| | coulomb_n.in | $U_{\alpha,\beta,\gamma,\delta}$ terms in $\hat{H}_n$ |
| | transop_rixs_i.in | transition operator $\hat{\mathcal{D}}_i$ |
| | transop_rixs_f.in | transition operator $\hat{\mathcal{D}}_f^\dagger$ |
| | eigvec.*i* | the *i*th ground state $\|\Gamma_i\rangle$ of $\hat{H}_i$ |
| Outputs | standard outputs | log file |
| | rixs_poles.*i* | RIXS data for the *i*th ground state |

**Table 5**
The approximate total running times of the three examples on a Macbook laptop.

| | Number of CPU cores | Total running time (s) |
|---|---|---|
| Example-1 (Section 5.1) | 1 | 5 |
| Example-2 (Section 5.2) | 4 | 10 |
| Example-3 (Section 5.3) | 4 | 3000 |

### 5.1. Hello RIXS: multiplets d-d excitations in a single atom case Ni ($d^8$)

In this example, we use pure Python code to do ED calculation and calculate XAS and RIXS spectra step by step. The main purpose is to show how the Python APIs are used. This example is a resonant x-ray scattering at Ni-$L_{2/3}$ edge ($2p_{1/2,3/2} \rightarrow 3d$ transition). Ni has a $d^8$ configuration with a tetragonal CF environment [3]. The Python codes are shown in the Listing 1.

**Listing 1:** A Python example to calculate XAS and RIXS

```
#!/usr/bin/env python

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator
import edrixs

font = {'family'  : 'Times New Roman' ,
        'weight' : 'medium' ,
        'size'   : '18'}
plt.rc('font',**font)

def ed():
    # 1-10: Ni-3d valence orbitals, 11-16: Ni-2p core orbitals
    # Single particle basis: complex shperical Harmonics
    ndorb, nporb, ntot = 10, 6, 16
    emat_i = np.zeros((ntot,ntot), dtype=np.complex)
    emat_n = np.zeros((ntot,ntot), dtype=np.complex)

    # 4-index Coulomb interaction tensor, parameterized by
    # Slater integrals, which are obtained from Cowan's code
    # Interaction among 3d valence electrons
    # averaged dd Coulomb interaction is set to be zero
    F2_d, F4_d = 7.9521,  4.9387
    F0_d = edrixs.get_F0('d', F2_d, F4_d)

    # Core-hole potential: interaction between 3d and 2p
    # averaged dp Coulomb interaction is set to be zero
    G1_dp, G3_dp = 4.0509, 2.3037
    F0_dp, F2_dp = edrixs.get_F0('dp', G1_dp, G3_dp), 7.33495

    umat_i = edrixs.get_umat_slater('dp', F0_d, F2_d, F4_d,  # dd
                                    0, 0, 0, 0,              # dp
                                    0, 0)                    # pp
    umat_n = edrixs.get_umat_slater('dp', F0_d, F2_d, F4_d,    # dd
                                    F0_dp, F2_dp, G1_dp, G3_dp,# dp
                                    0, 0)                      # pp
```

```
    # Atomic spin-orbit coupling
    zeta_d, zeta_p = 0.083, 11.24
    emat_i[0:ndorb, 0:ndorb] += edrixs.atom_hsoc('d', zeta_d)
    emat_n[0:ndorb, 0:ndorb] += edrixs.atom_hsoc('d', zeta_d)
    emat_n[ndorb:ntot, ndorb:ntot] += edrixs.atom_hsoc('p', zeta_p)

    # Tetragonal crystal field splitting terms,
    # which are first defined in the real cubic Harmonics basis,
    # and then transformed to complex shperical Harmonics basis.
    dt, ds, dq = 0.011428, 0.035714, 0.13
    tmp = np.zeros((5,5),dtype=np.complex)
    tmp[0,0] =  6*dq - 2*ds - 6*dt   # d3z2-r2
    tmp[1,1] = -4*dq - 1*ds + 4*dt   # dzx
    tmp[2,2] = -4*dq - 1*ds + 4*dt   # dzy
    tmp[3,3] =  6*dq + 2*ds - 1*dt   # dx2-y2
    tmp[4,4] = -4*dq + 2*ds - 1*dt   # dxy
    tmp[:,:] = edrixs.cb_op(tmp, edrixs.tmat_r2c('d'))
    emat_i[0:ndorb:2, 0:ndorb:2] += tmp
    emat_i[1:ndorb:2, 1:ndorb:2] += tmp
    emat_n[0:ndorb:2, 0:ndorb:2] += tmp
    emat_n[1:ndorb:2, 1:ndorb:2] += tmp

    # Build Fock basis in its binary form
    basis_i = edrixs.get_fock_bin_by_N(ndorb, 8, nporb, nporb)
    basis_n = edrixs.get_fock_bin_by_N(ndorb, 9, nporb, nporb-1)
    ncfg_i, ncfg_n = len(basis_i), len(basis_n)

    # Build many-body Hamiltonian in Fock basis
    hmat_i = np.zeros((ncfg_i,ncfg_i), dtype=np.complex)
    hmat_n = np.zeros((ncfg_n,ncfg_n), dtype=np.complex)
    hmat_i[:,:] += edrixs.two_fermion(emat_i,  basis_i, basis_i)
    hmat_i[:,:] += edrixs.four_fermion(umat_i, basis_i)
    hmat_n[:,:] += edrixs.two_fermion(emat_n,   basis_n, basis_n)
    hmat_n[:,:] += edrixs.four_fermion(umat_n, basis_n)

    # Do exact-diagonalization to get eigenvalues and eigenvectors
    eval_i, evec_i = np.linalg.eigh(hmat_i)
    eval_n, evec_n = np.linalg.eigh(hmat_n)

    # Build dipolar transition operators
    dipole = np.zeros((3, ntot, ntot), dtype=np.complex)
    T_abs = np.zeros((3, ncfg_n, ncfg_i), dtype=np.complex)
    T_emi = np.zeros((3, ncfg_i, ncfg_n), dtype=np.complex)
    tmp = edrixs.get_trans_oper('dp')
    for i in range(3):
        dipole[i, 0:ndorb, ndorb:ntot] = tmp[i]
        # First, in the Fock basis
        T_abs[i] = edrixs.two_fermion(dipole[i], basis_n, basis_i)
        # Then, transfrom to the eigenvector basis
        T_abs[i] = edrixs.cb_op2(T_abs[i], evec_n, evec_i)
        T_emi[i] = np.conj(np.transpose(T_abs[i]))

    return eval_i, eval_n, T_abs, T_emi

def xas(eval_i, eval_n, T_abs):
    gs = list(range(0, 3)) # three lowest eigenstates are used
    prob = edrixs.boltz_dist([eval_i[i] for i in gs], 300)
    off = 857.4  # offset of the energy of the incident x-ray
    Gam_c = 0.2  # core-hole life-time broadening
    pol = np.array([1.0, 1.0, 1.0])/np.sqrt(3.0) # isotropic
    omega = np.linspace(-10, 20, 1000)
    xas = np.zeros(len(omega), dtype=np.float)
    # Calculate XAS spectrum
    for i, om in enumerate(omega):
        for j in gs:
            F_mag = (T_abs[0,:,j]*pol[0] +
                     T_abs[1,:,j]*pol[1] +
                     T_abs[2,:,j]*pol[2] )

            xas[i] += prob[j]*np.sum(np.abs(F_mag)**2 *
                      Gam_c/np.pi/((om-(eval_n[:]-eval_i[j]))**2
```

```
                        + Gam_c**2))

    # plot XAS
    plt.figure()
    ax = plt.subplot(1,1,1)
    plt.ylim([0,0.6])
    plt.plot(omega+off, xas, '-', color="blue")
    ax.xaxis.set_major_locator(MultipleLocator(5))
    ax.xaxis.set_minor_locator(MultipleLocator(2.5))
    ax.yaxis.set_major_locator(MultipleLocator(0.1))
    ax.yaxis.set_minor_locator(MultipleLocator(0.05))
    plt.xlabel(r'Energy of incident photon (eV)')
    plt.ylabel(r'XAS Intensity (a.u.)')
    plt.text(852, 0.52, r'$L_{3}$', fontsize=20)
    plt.text(869.5, 0.15, r'$L_{2}$', fontsize=20)
    plt.text(846.5, 0.55, r'(a)', fontsize=25)
    plt.subplots_adjust(left=0.15, right=0.95, bottom=0.13,
                        top=0.95,  wspace=0.05, hspace=0.00)
    plt.savefig('xas_ni.pdf')

def rixs(eval_i, eval_n, T_abs, T_emi):
    gs=list(range(0, 3))
    prob = edrixs.boltz_dist([eval_i[i] for i in gs], 300)
    off=857.4
    phi, thin, thout = 0, 15/180.0*np.pi, 75/180.0*np.pi
    # Life-time broadening of the intermediate
    # (with a core-hole) and final states, respectively
    Gam_c, Gam = 0.2, 0.1
    pol = [(0,0), (0,np.pi/2.0)] # pi-pi and pi-sigma polarization
    omega = np.linspace(-5.9, -0.9, 100)
    eloss = np.linspace(-0.5, 5.0, 1000)
    rixs=np.zeros((len(pol),len(eloss),len(omega)),dtype=np.float)

    # Calculate RIXS
    for i, om in enumerate(omega):
        F_fi=edrixs.scattering_mat(eval_i,eval_n,T_abs[:,:,gs],
                                   T_emi,om,Gam_c)
        for j, (alpha, beta) in enumerate(pol):
            ei, ef=edrixs.dipole_polvec_rixs(thin,thout,phi,
                                             alpha,beta)
            F_mag=np.zeros((len(eval_i),len(gs)),dtype=np.complex)
            for m in range(3):
                for n in range(3):
                    F_mag[:,:]+=ef[m]*F_fi[m,n]*ei[n]
            for m in gs:
                for n in range(len(eval_i)):
                    rixs[j,:,i]+=(prob[m]*np.abs(F_mag[n,m])**2
                    *Gam/np.pi/((eloss-(eval_i[n]-eval_i[m]))**2
                                +Gam**2 ))

    # plot RIXS map
    plt.figure()
    ax = plt.subplot(1,1,1)
    a,b,c,d=min(omega)+off,max(omega)+off, min(eloss), max(eloss)
    plt.imshow(rixs[0]+rixs[1],extent=[a,b,c,d],
               origin='lower', aspect='auto',cmap='rainbow',
               interpolation='bicubic')
    ax.xaxis.set_major_locator(MultipleLocator(1))
    ax.xaxis.set_minor_locator(MultipleLocator(0.5))
    ax.yaxis.set_major_locator(MultipleLocator(1))
    ax.yaxis.set_minor_locator(MultipleLocator(0.5))
    plt.xlabel(r'Energy of incident photon (eV)')
    plt.ylabel(r'Energy loss (eV)')
    plt.text(851.6, 4.5, r'(b)', fontsize=25)
    plt.subplots_adjust(left=0.1, right=0.95, bottom=0.13,
                        top=0.95,  wspace=0.05, hspace=0.00)
    plt.savefig("rixs_map_ni.pdf")

if __name__ == "__main__":
    print("edrixs >>> ED ...")
    eval_i, eval_n, T_abs, T_emi = ed()
    print("edrixs >>> Done ! ")
```

```
    print("edrixs >>> Calculate XAS ...")
    xas(eval_i, eval_n, T_abs)
    print("edrixs >>> Done ! ")

    print("edrixs >>> Calculate RIXS ...")
    rixs(eval_i, eval_n, T_abs, T_emi)
    print("edrixs >>> Done ! ")
```

The calculated XAS spectrum and a RIXS map at $L_3$ edge are plotted in Fig. 4. More results can be found in Ref. [3].

### 5.2. Two-site Ir–Ir cluster: dimer excitations

In this section, we show an example of two-sites Ir–Ir cluster model, where two $IrO_6$ octahedras share their face, see Fig. 5. This type of structure can be found in materials, such as $Ba_5AlIr_2O_{11}$ [49,50] and the 6H-hexagonal oxides $Ba_3AB_2O_9$ ($A$=In, Y, Lu, Na and $B$=Ru, Ir) [51–53]. Such face sharing structure leads to strong hopping between these two Ir sites, and dimer excitations have been observed in experimental RIXS spectra and confirmed by a two-site model simulation [54]. Here, we show how to do the calculations step by step. We will simulate RIXS spectra at Ir-$L_3$ edge for the cases with and without hopping between these two Ir sites based on a $t_{2g}$ two-sites model. More details of the methods and the Hamiltonian of the on-site Coulomb interaction, spin–orbit coupling (SOC), trigonal crystal field and the hoppings between the two Ir sites can be found in Ref. [54].

We first go to the *examples* directory and run the Python script,

```
$ cd ${EDRIXS_DIR}/examples/cpc/two_site_cluster
$ ./get_inputs.py
```

to do the following things,

1. It first makes four directories: *ed* for performing ED calculation, *xas* for performing XAS calculation, *rixs_pp* for performing RIXS calculation with $\pi$-$\pi$ polarization and *rixs_ps* for performing RIXS calculation with $\pi$-$\sigma$ polarization.
2. Set the control parameters for *ed.x*, *xas.x* and *rixs.x* in file *config.in*,

```
# File: config.in
&control
num_val_orbs=12  ! Number of total valence orbitals (t2g): 2x6=12
num_core_orbs=12 ! Number of total core orbitals (2p): 2x6=12
ed_solver=0      ! Type of ED solver, full diagonalization of H_{i}
neval=220        ! Number of eigenvalues obtained
nvector=2      ! Number of eigenvectors obtained
idump=.true.     ! Dump eigenvectors to file eigvec.xxx
num_gs=2       ! Numer of gound states used for XAS and RIXS
           ! calculations
linsys_tol=1E-10  ! Tolerance for the termination of solving
           ! linear equations
nkryl=500      ! Maximum number of Krylov vectors when building
           ! Krylov space
gamma_in=2.5    ! Core-hole life-time in eV
omega_in=-540.4  ! Incident x-ray energy at which RIXS
           ! calculations are performed
&end
```

3. Get parameters of hopping and Coulomb interaction $t_{\alpha,\beta}$ and $U_{\alpha,\beta,\gamma,\delta}$, and write them in files: *hopping_i.in*, *hopping_n.in*, *coulomb_i.in* and *coulomb_n.in*.
4. Get Fock basis and write to files: *fock_i.in*, *fock_n.in* and *fock_f.in*. Please note that when setting up the Fock basis, we only consider the valence orbitals because the core orbitals will be explicitly considered within *ed.x*, *xas.x* and *rixs.x*. Here, for $\hat{H}_i$, there will be 9 electrons occupying 12 valence orbitals, and for $\hat{H}_n$ with a core-hole, there will be 10 electrons occupying 12 valence orbitals.
5. Get transition operators for XAS and RIXS and write to files: *transop_xas.in*, *transop_rixs_i.in* and *transop_rixs_f.in*. For XAS, we use isotropic polarization. For RIXS, we do calculations for $\pi$-$\pi$ and $\pi$-$\sigma$ polarization. The scattering plane is chosen as $ac$-plane. The incident and scattered angles are $\theta_{\text{in}} = \pi/6$ and $\theta_{\text{out}} = \pi/3$, respectively.

Then, we go to the *ed* directory and run

```
$ mpirun -np 4 ed.x > log.dat
```

to perform the ED calculation. After ED is finished, we copy eigenvectors to *xas*, *rixs_pp* and *rixs_ps* directories,

```
$ cp eigvec.* ../xas
$ cp eigvec.* ../rixs_pp
$ cp eigvec.* ../rixs_ps
```

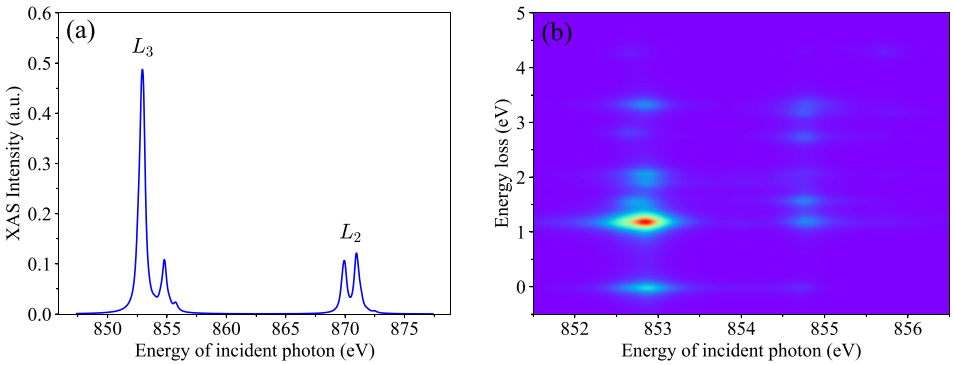


**Fig. 4.** (a) XAS spectrum of Ni; (b) a RIXS map at $L_3$ edge.

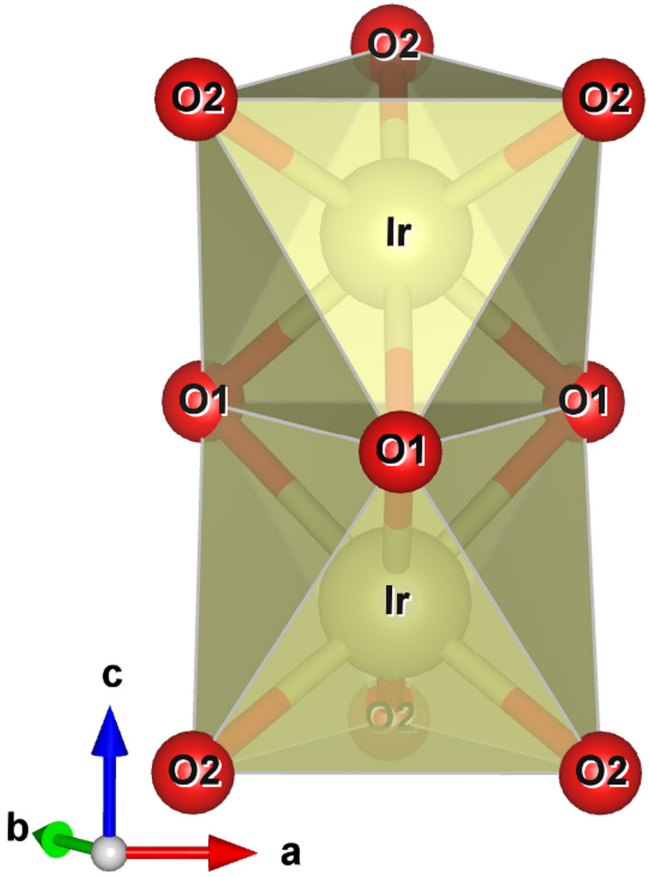


**Fig. 5.** The illustration of the face sharing of two $IrO_6$ octahedras.

We can now go to *xas* directory to do XAS calculation and just type

```
$ mpirun -np 4 xas.x > log.dat
```

After the XAS calculation is finished, we can get the XAS spectrum by running a Python script in upper directory

```
$ get_spectrum.py -N 1000 -ommin -560 -ommax -520 -T 300 -G 2.5 \
> -off 11755.5 -f xas_with_hopp.dat xas/*poles*
```

where we get the XAS spectrum near Ir-$L_3$ edge among energy range [ommin+off,ommax+off]. Temperature is set to be 300 K and $\Gamma_c = 2.5$ eV. We can now plot the XAS spectrum, which is shown in Fig. 6(a).

Finally, we go to the *rixs_pp* and *rixs_ps* directories to do RIXS calculations. We need manually change *omega_in* in file *config.in* to set the incident x-ray energy at which the RIXS are performed. Here, we set it to be $-540.4$ eV, which is corresponding to the resonant energy 11.215 keV of the Ir-$L_3$ edge. We run the command,

```
$ mpirun -np 4 rixs.x > log.dat
```

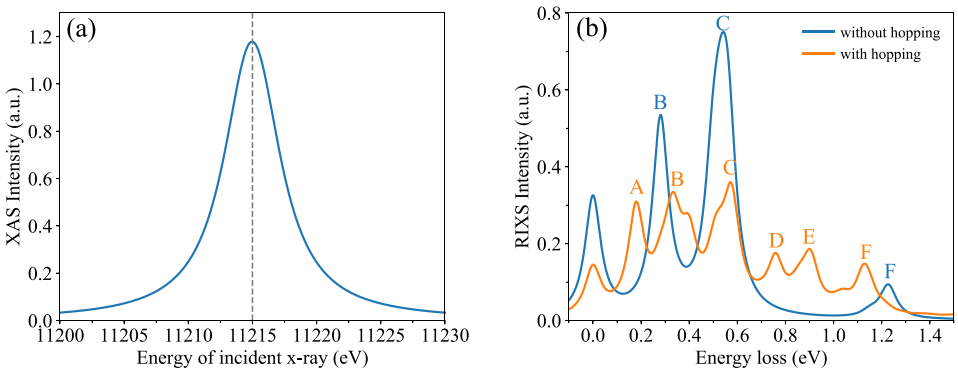


**Fig. 6.** (a) Simulated XAS spectrum at Ir-$L_3$ edge and (b) RIXS spectrum for the cases with and without hoppings between the two Ir-sites.

to launch the RIXS calculations. After the calculations are finished, we run the Python script

```
$ get_spectrum.py -N 1000 -ommin -0.2 -ommax 1.5 -T 300 -G 0.04  \
> -f rixs_with_hopp.dat rixs_pp/*poles* rixs_ps/*poles*
```

to get the RIXS spectrum among energy range [-0.2, 1.5]. $\Gamma$ is set to be 0.04 eV. We repeat the above calculations for the case without hoppings between the two Ir sites by setting $t_1 = t_2 = 0.0$ in file *get_inputs.py*. The RIXS spectrum is plotted in Fig. 6(b). Without hoppings, we can find three peaks B, C, and F corresponding to the local *d-d* excitations which are mainly determined by Hund's coupling $J_H$ and SOC $\lambda$. After turning on the hoppings, the peak positions of these *d-d* excitations will be shifted and new peaks A, D and E corresponding to dimer excitations appear in the RIXS spectrum. The onset of dimer excitations in RIXS spectrum has been observed in $Ba_5AlIr_2O_{11}$ experimentally [54]. Here, we demonstrate that EDRIXS can be helpful to simulate such dimer excitations in materials.

### 5.3. Anderson impurity model: charge transfer excitations

In this section, we show an example to simulate RIXS spectrum based on Anderson impurity model in an Os compound [55] with nominal $d^3$ configuration to get charge transfer excitations. We use a $t_{2g}$ Anderson impurity model with three bath sites per orbitals. The bath sites are obtained from a converged DFT+DMFT calculations. We first go to the *examples* directory and run the Python script,

```
$ cd ${EDRIXS_DIR}/examples/cpc/anderson_impurity
$ ./get_inputs.py
```

to build necessary directories, prepare input files just like what we do in Section 5.2. For Anderson impurity model, we first need to perform ED calculations in subspaces with fixed total occupancy number to determine the total occupancy number of the ground states.

```
$ cd search_gs
$ echo "mpirun -np 4 ${EDRIXS_DIR}/bin/ed.x" > mpi_cmd.dat
$ ${EDRIXS_DIR}/bin/search_gs_by_N.py -ntot 24 -nimp 6  \
> -N1 2 -N2 22 -mpi_cmd mpi_cmd.dat
$ cd ..
```

where the total number of orbitals is 24 and the first 6 are the impurity orbitals. We perform the ED calculations for the total occupancy number ranging from 2 to 22. For these ED calculations, we only need to get rough results, so we use the ED solver with standard Lanczos algorithm without re-orthogonalization by setting *ed_solver=1* in file *config.in*.

After the ED calculations are done, we find that the total occupancy of the global ground state is 15 in the first line of the file *results.dat*. Then, we run

```
$ ./get_fock_basis.py -norb 24 -noccu 15
```

to set up Fock basis. After that, we can go to the *ed* directory to perform an ED calculation with total occupancy 15. At this time, we use the ED solver with parallel ARPACK library to get more accurate ground states by setting *ed_solver=2*.

```
$ cd ed
$ mpirun -np 4 ${EDRIXS_DIR}/bin/ed.x > log.dat
$ cp eigvec.* ../xas
$ cp eigvec.* ../rixs_pp
$ cp eigvec.* ../rixs_ps
$ cd ..
```

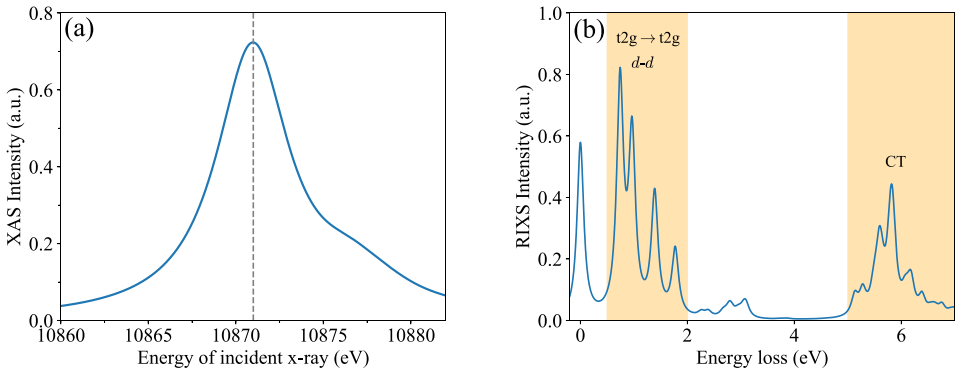


**Fig. 7.** (a) Simulated XAS spectrum and (b) RIXS spectrum of an Os compound based on an Anderson impurity model.

We go to the *xas* directory to perform a XAS calculation and then get the XAS spectrum by running,

```
$ get_spectrum.py -N 1000 -ommin -20 -ommax 10 -T 50 -G 2.5  \
> -off 10877.2 -f xas.dat xas/*poles*
```

Finally, we can perform RIXS calculations for polarization $\pi$-$\pi$ and $\pi$-$\sigma$. We first set *omega_in* to be $-6.2$, which is corresponding to the resonant energy 10871 eV at Os-$L_3$ edge.

```
$ cd rixs_pp
$ mpirun -np 4 ${EDRIXS_DIR}/bin/rixs.x > log.dat
$ cd ../rixs_ps
$ mpirun -np 4 ${EDRIXS_DIR}/bin/rixs.x > log.dat
$ cd ..
```

After that, we run

```
$ ${EDRIXS_DIR}/bin/get_spectrum.py -N 1000 -ommin -0.2 -ommax 7  \
> -T 50 -G 0.075 -f rixs.dat rixs_pp/*poles* rixs_ps/*poles*
```

to get the simulated RIXS spectrum.

The simulated XAS and RIXS spectrum are plotted in Fig. 7. In Fig. 7(b), we can see four peaks below 2 eV, which are corresponding to *d*-*d* excitations in the $t_{2g}$ subspace, and a broad charge transfer peak in the range 5–7 eV. More details and results of the RIXS simulations based on DFT+DMFT calculations for this compound will be published in another separate paper [56].

## 6. Summary and future developments

In this paper, we introduce the open source toolkit EDRIXS to simulate RIXS spectra. We explain the basic theory and algorithms for RIXS simulations and the implementation details of EDRIXS code. We also show three examples to demonstrate its usage. EDRIXS is designed as a platform for theoretical simulations of x-ray scattering spectroscopy and it will be a very helpful toolkit for the x-ray scattering community.

The development of EDRIXS code is still in progress. The plans of future development are likely to be along the following directions. More powerful ED solvers based on algorithms such as quantum chemistry (configuration interaction) [57–60], NRG with non-abelian symmetries [61] will be implemented to diagonalize even larger size of Anderson impurity model. The Python API will be enhanced to provide more powerful and friendly functionalities. A database for Slater integrals of atoms will be provided. Graphical user interface (GUI) can also be implemented for easy use, especially for the experimentalists at beam lines.

## Acknowledgments

We would like to thank very helpful discussions with Hu Miao and Yongxin Yao. Y.L.W., M.P.M.D. and G.K. were supported by the US Department of energy, Office of Science, Basic Energy Sciences, USA as a part of the Computational Materials Science Program through the Center for Computational Design of Functional Strongly Correlated Materials and Theoretical Spectroscopy. G.F. was supported by the U.S. Department of Energy, Office of Basic Energy Sciences, Early Career Award Program under Award No. 1047478. Work at Brookhaven National Laboratory was supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under Contract No. DE-SC0012704.

## References


[1] A. Damascelli, Z. Hussain, Z.-X. Shen, Rev. Modern Phys. 75 (2003) 473–541.
[2] L.J.P. Ament, M. van Veenendaal, T.P. Devereaux, J.P. Hill, J. van den Brink, Rev. Modern Phys. 83 (2) (2011) 705–767.
[3] G. Fabbris, D. Meyers, J. Okamoto, J. Pelliciari, A.S. Disa, Y. Huang, Z.-Y. Chen, W.B. Wu, C.T. Chen, S. Ismail-Beigi, C.H. Ahn, F.J. Walker, D.J. Huang, T. Schmitt, M.P.M. Dean, Phys. Rev. Lett. 117 (2016) 147401.
[4] K. Tomiyasu, J. Okamoto, H.Y. Huang, Z.Y. Chen, E.P. Sinaga, W.B. Wu, Y.Y. Chu, A. Singh, R.-P. Wang, F.M.F. de Groot, A. Chainani, S. Ishihara, C.T. Chen, D.J. Huang, Phys. Rev. Lett. 119 (2017) 196402.
[5] J. Kim, D. Casa, M.H. Upton, T. Gog, Y.-J. Kim, J.F. Mitchell, M. van Veenendaal, M. Daghofer, J. van den Brink, G. Khaliullin, B.J. Kim, Phys. Rev. Lett. 108 (2012) 177003.
[6] L.A. Wray, J. Denlinger, S.-W. Huang, H. He, N.P. Butch, M.B. Maple, Z. Hussain, Y.-D. Chuang, Phys. Rev. Lett. 114 (2015) 236401.
[7] A.M. Shvaika, O. Vorobyov, J.K. Freericks, T.P. Devereaux, Phys. Rev. B 71 (2005) 045120.
[8] J.-i. Igarashi, T. Nagao, Phys. Rev. B 88 (2013) 014407.
[9] A. Georges, G. Kotliar, W. Krauth, M.J. Rozenberg, Rev. Modern Phys. 68 (1996) 13–125.
[10] G. Kotliar, S.Y. Savrasov, K. Haule, V.S. Oudovenko, O. Parcollet, C.A. Marianetti, Rev. Modern Phys. 78 (2006) 865–951.
[11] E. Gull, A.J. Millis, A.I. Lichtenstein, A.N. Rubtsov, M. Troyer, P. Werner, Rev. Modern Phys. 83 (2011) 349–404.
[12] F. De Groot, A. Kotani, Core Level Spectroscopy of Solids, in: Advances in Condensed Matter Science, Taylor & Francis Group, 2008.
[13] E. Stavitski, F.M. de Groot, Micron 41 (7) (2010) 687–694.
[14] M.W. Haverkort, J. Phys. Conf. Ser. 712 (1) (2016) 012001.
[15] R.D. Cowan, The Theory of Atomic Structure and Spectra, University of California Press, Berkeley, 1981.
[16] https://www.lua.org/.
[17] K. Tsutsui, T. Tohyama, S. Maekawa, Phys. Rev. Lett. 91 (2003) 117001.
[18] K. Ishii, K. Tsutsui, Y. Endoh, T. Tohyama, S. Maekawa, M. Hoesch, K. Kuzushita, M. Tsubota, T. Inami, J. Mizuki, Y. Murakami, K. Yamada, Phys. Rev. Lett. 94 (2005) 207003.
[19] K. Okada, A. Kotani, J. Phys. Soc. Japan 75 (4) (2006) 044702.
[20] F. Vernay, B. Moritz, I.S. Elfimov, J. Geck, D. Hawthorn, T.P. Devereaux, G.A. Sawatzky, Phys. Rev. B 77 (2008) 104519.
[21] C.-C. Chen, B. Moritz, F. Vernay, J.N. Hancock, S. Johnston, C.J. Jia, G. Chabot-Couture, M. Greven, I. Elfimov, G.A. Sawatzky, T.P. Devereaux, Phys. Rev. Lett. 105 (2010) 177401.
[22] S. Kourtis, J. van den Brink, M. Daghofer, Phys. Rev. B 85 (2012) 064423.
[23] A. Uldry, F. Vernay, B. Delley, Phys. Rev. B 85 (2012) 125133.
[24] C. Jia, C.-C. Chen, A. Sorini, B. Moritz, T. Devereaux, New J. Phys. 14 (11) (2012) 113038.
[25] K. Wohlfeld, S. Nishimoto, M.W. Haverkort, J. van den Brink, Phys. Rev. B 88 (2013) 195138.
[26] C. Monney, V. Bisogni, K.-J. Zhou, R. Kraus, V.N. Strocov, G. Behr, J. Málek, R. Kuzian, S.-L. Drechsler, S. Johnston, A. Revcolevschi, B. Büchner, H.M. Rønnow, J. van den Brink, J. Geck, T. Schmitt, Phys. Rev. Lett. 110 (2013) 087403.
[27] C.J. Jia, E.A. Nowadnick, K. Wohlfeld, Y.F. Kung, C.-C. Chen, S. Johnston, T. Tohyama, B. Moritz, T.P. Devereaux, Nature Commun. 5 (2014) 3314 EP, Article.
[28] J. Fernndez-Rodríguez, B. Toby, M. van Veenendaal, J. Electron Spectrosc. Relat. Phenom. 202 (2015) 81–88.
[29] C. Jia, K. Wohlfeld, Y. Wang, B. Moritz, T.P. Devereaux, Phys. Rev. X 6 (2016) 021020.
[30] K. Tsutsui, T. Tohyama, Phys. Rev. B 94 (2016) 085144.
[31] R.J. Green, M.W. Haverkort, G.A. Sawatzky, Phys. Rev. B 94 (2016) 195127.
[32] B. Yuan, J. Clancy, A. Cook, C. Thompson, J. Greedan, G. Cao, B. Jeon, T. Noh, M. Upton, D. Casa, et al., Phys. Rev. B 95 (23) (2017) 235114.
[33] B.J. Kim, G. Khaliullin, Phys. Rev. B 96 (2017) 085108.
[34] A. Paramekanti, D.J. Singh, B. Yuan, D. Casa, A. Said, Y.-J. Kim, A.D. Christianson, Phys. Rev. B 97 (2018) 235119.
[35] A. Hariki, M. Winder, J. Kuneš, Phys. Rev. Lett. 121 (2018) 126403.
[36] U. Kumar, A. Nocera, E. Dagotto, S. Johnston, New J. Phys. 20 (7) (2018) 073019.
[37] J. Schlappa, U. Kumar, K.J. Zhou, S. Singh, M. Mourigal, V.N. Strocov, A. Revcolevschi, L. Patthey, H.M. Rønnow, S. Johnston, T. Schmitt, Nat. Commun. 9 (1) (2018) 5394.
[38] A. Nocera, U. Kumar, N. Kaushal, G. Alvarez, E. Dagotto, S. Johnston, Sci. Rep. 8 (1) (2018) 11080.
[39] https://www.bnl.gov/comscope/index.php.
[40] S. Choi, P. Semon, B. Kang, A. Kutepov, G. Kotliar, ComDMFT: a massively parallel computer package for the electronic structure of correlated-electron systems, 2018, arXiv:1810.01679.
[41] C. Lanczos, J. Res. Natl. Bur. Stand. B 45 (1950) 255–282.
[42] W.E. Arnoldi, Quart. Appl. Math. 9 (1951) 17–29.
[43] D.C. Sorensen, in: D.E. Keyes, A. Sameh, V. Venkatakrishnan (Eds.), Parallel Numerical Algorithms, Springer Netherlands, Dordrecht, 1997, pp. 119–165.
[44] http://web.stanford.edu/group/SOL/software/minres/.
[45] https://www.tcd.ie/Physics/people/Cormac.McGuinness/Cowan/.
[46] https://www.anaconda.com/.
[47] https://www.openblas.net/.
[48] https://github.com/opencollab/arpack-ng.
[49] J. Terzic, J. Wang, F. Ye, W. Song, S. Yuan, S. Aswartham, L.E. DeLong, S. Streltsov, D.I. Khomskii, G. Cao, Phys. Rev. B 91 (23) (2015) 235147.
[50] S.V. Streltsov, G. Cao, D.I. Khomskii, Phys. Rev. B 96 (2017) 014434.
[51] T. Dey, M. Majumder, J.C. Orain, A. Senyshyn, M. Prinz-Zwick, S. Bachus, Y. Tokiwa, F. Bert, P. Khuntia, N. Büttgen, A.A. Tsirlin, P. Gegenwart, Phys. Rev. B 96 (2017) 174411.
[52] D. Ziat, A.A. Aczel, R. Sinclair, Q. Chen, H.D. Zhou, T.J. Williams, M.B. Stone, A. Verrier, J.A. Quilliam, Phys. Rev. B 95 (2017) 184424.
[53] S.A.J. Kimber, M.S. Senn, S. Fratini, H. Wu, A.H. Hill, P. Manuel, J.P. Attfield, D.N. Argyriou, P.F. Henry, Phys. Rev. Lett. 108 (2012) 217205.
[54] Y. Wang, R. Wang, J. Kim, M.H. Upton, D. Casa, T. Gog, G. Cao, G. Kotliar, M.P.M. Dean, X. Liu, Phys. Rev. Lett. 122 (2019) 106401.
[55] A.E. Taylor, S. Calder, R. Morrow, H.L. Feng, M.H. Upton, M.D. Lumsden, K. Yamaura, P.M. Woodward, A.D. Christianson, Phys. Rev. Lett. 118 (2017) 207202.
[56] The manuscript is under preparation.
[57] K.R. Shamasundar, G. Knizia, H.-J. Werner, J. Chem. Phys. 135 (5) (2011) 054101.
[58] D. Zgid, E. Gull, G.K.-L. Chan, Phys. Rev. B 86 (2012) 165128.
[59] C. Lin, A.A. Demkov, Phys. Rev. B 88 (2013) 035123.
[60] Y. Lu, M. Höppner, O. Gunnarsson, M.W. Haverkort, Phys. Rev. B 90 (2014) 085102.
[61] A. Weichselbaum, Ann. Physics 327 (12) (2012) 2972–3047.