Cif2x Documentation

Release 1.0-alpha

ISSP, University of Tokyo

Dec 26, 2023

Contents

Chapter 1

Introduction

1.1 What is cif2x?

In recent years, the use of machine learning for predicting material properties and designing substances (known as materials informatics) has gained considerable attention. The accuracy of machine learning depends heavily on the preparation of appropriate training data. Therefore, the development of tools and environments for the rapid generation of training data is expected to contribute significantly to the advancement of research in materials informatics.

Cif2x is a tool that generates input files for first-principles calculations from cif files. It constructs parts that vary depending on the type of material and computational conditions from crystal structure data, using input parameters as a template. It is capable of generating multiple input files tailored to specific computational conditions. Currently, it supports *VASP <https://www.vasp.at>*, *Quantum ESPRESSO <https://www.quantum-espresso.org>*, and *OpenMX <http://www.openmx-square.org>*, with plans to support *AkaiKKR <http://kkr.issp.u-tokyo.ac.jp>* in the future.

1.2 License

The distribution of the program package and the source codes for cif2x follow GNU General Public License version 3 (GPL v3) or later.

1.3 Contributors

This software was developed by the following contributors.

- ver.1.0-alpha (Released on 2023/12/28)
  - Developers
    - Kazuyoshi Yoshimi (The Instutite for Solid State Physics, The University of Tokyo)
    - Tatsumi Aoyama (The Instutite for Solid State Physics, The University of Tokyo)
    - Yuichi Motoyama (The Instutite for Solid State Physics, The University of Tokyo)
    - Masahiro Fukuda (The Instutite for Solid State Physics, The University of Tokyo)
    - Tetsuya Fukushima (The National Institute of Advanced Industrial Science and Technology (AIST))
    - Kota Ido (The Instutite for Solid State Physics, The University of Tokyo)

        * Shusuke Kasamatsu (Yamagata University)
        * Takashi Koretsune (Tohoku University)
    - Project Corrdinator
        * Taisuke Ozaki (The Instutite for Solid State Physics, The University of Tokyo)

1.4 Copyright

This software was developed with the support of " Project for advancement of software usability in materials science " of The Institute for Solid State Physics, The University of Tokyo.

1.5 Operating environment

This tool was tested on the following platforms:

- Ubuntu Linux + python3

Chapter 2

Installation and basic usage

Prerequisite

Input file generator for first-principles calculation `cif2x` included in HTP-tools requires the following programs and libraries:

- python 3.x
- pymatgen module
- ruamel.yaml module
- f90nml module
- qe-tools module
- numpy module
- pandas module
- monty module
- OpenBabel module (optional)

Official pages

- GitHub repository

Downloads

cif2x can be downloaded by the following command with git:

```
$ git clone https://github.com/issp-center-dev/cif2x.git
```

Installation

Once the source files are obtained, you can install cif2x by running the following command. The required libraries will also be installed automatically at the same time.

```
$ cd ./cif2x
$ python3 -m pip install .
```

The executable file `cif2x` will be installed. You may need to add `--user` option next to `install` keyword above in case you are not allowed to install packages system-wide.

Directory structure

```
.
|-- LICENSE
|-- README.md
|-- pyproject.toml
|-- docs/
|   |-- ja/
|   |-- en/
|   |-- tutorial/
|-- src/
|   |-- cif2x/
|       |-- __init__.py
|       |-- main.py
|       |-- cif2struct.py
|       |-- struct2qe.py
|       |-- qe/
|       |   |-- __init__.py
|       |   |-- calc_mode.py
|       |   |-- cards.py
|       |   |-- content.py
|       |   |-- qeutils.py
|       |   |-- tools.py
|       |-- struct2vasp.py
|       |-- struct2openmx.py
|       |-- openmx/
|       |   |-- __init__.py
|       |   |-- vps_table.py
|       |-- utils.py
|-- sample/
```

Basic usage

`cif2x` is a tool to generate a set of input files for first-principles calculation software. It takes an input parameter file as a template, and generates parameter items that may vary by materials and calculation conditions from crystallographic data. In the present version, `cif2x` supports Quantum ESPRESSO, VASP, and OpenMX.

1. Prepare input parameter file

   First, you need to create an input parameter file in YAML format that describes contents of the input file to be generated for the first-principles calculation software.

2. Prepare crystal structure files and pseudo-potential files

   The crystal structure data need to be prepared for the target materials. The file format is CIF, POSCAR, xfs, or those supported by pymatgen.

   For Quantum ESPRESSO, the pseudo-potential files and the index file in CSV format need to be placed. Their locations are specified in the input parameter file.

   For VASP, the location of the pseudo-potential files will be specified in a file `~/.config/.pmgrc.yaml` or by an environment variable. It may be specified in the input parameter file.

3. Run command

   Run `cif2x` command with the input parameter file and the crystal structure data as arguments. To generate input files for Quantum ESPRESSO, the target option `-t QE` should be

specified. The option turns to `-t VASP` for VASP, and `-t OpenMX` for OpenMX.

```
$ cif2x -t QE input.yaml material.cif
```

Chapter 3

Tutorial

The procedure to use the input file generator `cif2x` for first-principles calculation software consists of preparing an input parameter file, crystal structure data, and pseudo-potential files, and running the program `cif2x`. In the current version, the supported software includes Quantum ESPRESSO, VASP, and OpenMX. In this tutorial, we will explain the steps along a sample for Quantum ESPRESSO in `docs/tutorial/cif2x`.

3.1 Prepare an input parameter file

An input parameter file describes the content of input files for the first-principles calculation software. An example is presented below. It is a text file in YAML format that contains options to crystal structure data, and contents of the input file used as an input for the first-principle calculation. See *file format* section for the details of specification.

In YAML format, parameters are given in dictionary form as `keyword: value`, where `value` is a scalar such as a number or a string, or a set of values enclosed in `[ ]` or listed in itemized form, or a nested dictionary.

```
structure:
  use_ibrav: false
  tolerance: 0.05

optional:
  pseudo_dir: ./pseudo
  pp_file: ./pseudo/pp_psl_pbe_rrkjus.csv

tasks:
  - mode: scf
    output_file: scf.in
    output_dir: scf
    template: scf.in_tmpl
    content:
      namelist:
        control:
          prefix: pwscf
          pseudo_dir:
          outdir: ./work
        system:
          ecutwfc:
```

(continues on next page)

(continued from previous page)

```
      ecutrho:
    CELL_PARAMETERS:
    ATOMIC_SPECIES:
    ATOMIC_POSITIONS:
      option: crystal
    K_POINTS:
      option: automatic
      grid: [8,8,8]
```

The input parameter file consists of `structure`, `optional`, and `tasks` sections. The `structure` section specifies options to the crystal structure data. The `optional` section holds global settings concerning the pseudo-potentials.

The `tasks` section describes inputs for the first-principles calculations. In case of generating multiple files for a series of calculations, the `tasks` section takes a list of parameter sets. For each set, the calculation type is specified by the `mode` parameter: `scf` and `nscf` are supported as modes, as well as arbitrary modes for generic output.

The content of the output is given in `content` section. The input files of Quantum ESPRESSO are composed of the parts in namelist format of Fortran90 starting from `&keyword`, and the blocks called cards that start with keywords such as `K_POINTS` and end with blank lines. The `content` block holds namelist and cards in a form of nested dictionary. Basically, the specified items are written to the input files as-is, except for several cases. If a keyword is left blank, its value will be obtained form the crystal structure data or other sources.

Besides, templates of the input files can be used. The content of the file given by the `template` keyword is considered as input data along with the entries in `content` block. When the entries of the same keywords appear both, those of the input parameter files will be used. Therefore, it is possible to use template files and overwrite some entries by the input parameter file as needed. In the present example, the file (`scf.in_tmpl`) shown below is read as a template, and the entries on cutoff parameters as well as cards of CELL_PARAMETER, ATOMIC_SPECIES, ATOMIC_POSITIONS, K_POINTS are generated from the crystal structure data and pseudo-potential files. It is noted that the values of `ecutwfc` and `ecutrho` are overwritten by the empty lines.

```
&control
    calculation = 'scf'
    prefix = 'pwscf'
    pseudo_dir = './pseudo'
    outdir = './work'
    tstress = .true.
    tprnfor = .true.
/

&system
    ibrav = 0
    nat = 7
    ntyp = 3
    ecutwfc = 36.0
    ecutrho = 180.0
    occupations = 'smearing'
    smearing = 'm-p'
    degauss = 0.01
    noncolin = .true.
    nspin = 2
/

&electrons
```

(continues on next page)

(continued from previous page)

```
    missing_beta = 0.1
    conv_thr = 1e-08
/
```

3.2 Generating input files

The program `cif2x` is executed with the input parameter file (`input.yaml`) and crystal structure data (`Co3SnS2_nosym.cif`) as follows.

```
$ cif2x -t QE input.yaml Co3SnS2_nosym.cif
```

The required pseudo-potential files should be placed in the directory `./pseudo`, and the index file for the pseudo-potential should be prepared as `./psudo/pp_psl_pbe_rrkjus.csv`.

Run `cif2x` and a set of input files for Quantum ESPRESSO will be created. The output file is specified by `output_file` parameter of the input parameter file, and stored in the directory given by `output_dir`. In this example, the input file for SCF calculation is created as `./scf/scf.in`.

3.3 Specifying parameter sets

In some cases, a series of input files should be generated with varying their parameter values. For example, the convergence is examined by modifying the cutoff values or grid resolution of k points. The input parameter can be given a list or a range of values, and the input files for every combination from the choices of parameter values are generated and stored in separate directories. To specify parameter set, a special syntax `${...}` is adopted.

```
content:
  K_POINTS:
    option: automatic
    grid:   ${ [ [4,4,4], [8,8,8], [12,12,12] ] }
```

When `K_POINTS` is given as above, the input files having the `grid` value to be `[4,4,4]`, `[8,8,8]`, `[12,12,12]` will be generated in the sub-directories, `4x4x4/`, `8x8x8/`, `12x12x12/`, respectively.

Chapter 4

Command reference

4.1 cif2x

Generate input files for first-principles calculation software

SYNOPSIS:

```
cif2x [-v][-q] -t target input_yaml material.cif
cif2x -h
cif2x --version
```

DESCRIPTION:

This program reads an input parameter file specified by `input_yaml` and a crystal data file specified by `material.cif`, and generates a set of input files for first-principles calculation software. In the current version, the supported software includes Quantum ESPRESSO, VASP, and OpenMX. It takes the following command line options.

- `-v`

  increases verbosity of the runtime messages. When specified multiple times, the program becomes more verbose.

- `-q`

  decreases verbosity of the runtime messages. It cancels the effect of `-v` option, and when specified multiple times, the program becomes more quiet.

- `-t` *target*

  specifies the target first-principles calculation software. The supported software for *target* is listed as follows:

  - `QE`, `espresso`, `quantum_espresso`: generates input files for Quantum ESPRESSO.
  - `VASP`: generates input files for VASP.
  - `OpenMX`: generates input files for OpenMX

- `input_yaml`

  specifies an input parameter file in YAML format.

- `material.cif`

  specifies crystal structure data file. It is in CIF (Crystallographic Information Framework) format, or other format supported by pymatgen.

- `-h`

  displays help and exits.

- `--version`

  displays version information.

Chapter 5

File format

5.1 Input parameter file

An input parameter file describes information necessary to generate input files for first-principles calculation software by `cif2x`. It should be given in YAML format, and consist of the following sections.

1. structure section: describes how to handle crystal structure data.
2. optional section: describes pseudo-potential files, and symbol definitions for reference feature of YAML.
3. tasks section: describes contents of input files.

5.1.1 structure

`use_ibrav` (default value: `false`)

This parameter specifies whether `ibrav` parameter is used for Quantum ESPRESSO as the input of the crystal structure. When it is set to `true`, the lattice is transformed to match the convention of Quantum ESPRESSO, and the lattice parameters `a`, `b`, `c`, `cosab`, `cosac`, and `cosbc` are written to the input file as needed.

`tolerance` (default value: 0.01)

This parameter specifies the tolerance in the difference between the reconstructed Structure data and the original data when `use_ibrav` is set to `true`.

`supercell` (default value: none)

This parameter specifies the size of supercell, when it is adopted, in the form of $[n_x, n_y, n_z]$.

5.1.2 optional

This section contains global settings needed for the first-principles calculation software. The available parameters are described in the corresponding sections below.

5.1.3 tasks

This section defines contents of the input files. It is organized as a list of blocks, each corresponding to an input file, to allow for generating a set of input files for an input. The terms described in each block are explained in the following.

`mode` (Quantum ESPRESSO)

> This parameter specifies the type of calculation. In the current version, the supported mode includes `scf` and `nscf` for pw.x of Quantum ESPRESSO. If an unsupported mode is specified, the settings in `content` will be exported as is.

`output_file` (Quantum ESPRESSO)

> This parameter specifies the file name of the output.

`output_dir`

> This parameter specifies the directory name of the output. The default value is the current directory.

`content`

> This parameter describes the content of the output. For Quantum ESPRESSO, it contains the namelist data (blocks starting from `&system`, `&control`, etc.) in `namelist` block, and other card data (such as `K_POINTS`) as individual blocks. Some card data may take parameters.

`template` (Quantum ESPRESSO)

`template_dir` (VASP)

> These parameters specifies the template file and the template directory for the input files, respectively. If they are not given, templates will not be used. The content of the template file is merged with those of `content`. The entries in the template file will be superseded by those of `content` if the entries of the same keys appear both.

5.1.4 Specifying parameter set

An input parameter may be given a list or range of parameters. In this case, a separate directory is created for every combination of parameters to store the generated input files. A special syntax `${...}` is used to specify the parameter set as follows:

- a list: `${[ A, B, ... ]}`

  a set of parameter values is described as a Python list. Each entry may be a scalar value, or a list of values.

- a range: `${range(N)}`, `${range(start, end, step)}`

  a range of parameter is given by the keyword `range`. The former specifies the values from `0` to `N-1`, and the latter from `start` to `end` with every `step`. (If `step` is omitted, it is assumed to be `1`.)

5.2 Parameters for Quantum ESPRESSO

The entries of `optional` section and `content` part of the `tasks` section specific to Quantum ESPRESSO are explained below. In the current version, `scf` mode and `nscf` mode of `pw.x` are supported.

5.2.1 optional section

`pp_file`

This parameter specifies the index file in CSV format that relates the element type and the pseudo-potential file. This file contains the following columns: element name, type of pseudo-potential, nexclude, orbitals. An example line is given as:

```
Fe,pbe-spn-rrkjus_psl.0.2.1,4,spd
```

The name of the pseudo-potential file corresponding to the above example reads Fe.pbe-spn-rrkjus_psl.0.2.1.UPF .

`cutoff_file`

This parameter specifies the index file in CSV format that relates the pseudo-potential file and the cutoff values. This file contains the following columns: name of pseudo-potential file, `ecutwfc` value, `ecutrho` value.

`pseudo_dir`

This parameter specifies the name of the directory that holds pseudo-potential files. It is used when the cutoff values are obtained from the pseudo-potential files. It is indenepent from the `pseudo_dir` parameter in the input files for Quantum ESPRESSO.

5.2.2 content

namelist

- The lattice specifications in `&system` block will be superseded according to `use_ibrav` parameter in the `structure` section.
  - `use_ibrav = false`: `ibrav` is set to `0`, and the lattice parameters including `a`, `b`, `c`, `cosab`, `cosac`, `cosbc`, `celldm` are removed.
  - `use_ibrav = true`: `ibrab` is set to the index of Bravais lattices obtained from the crystal structure data. The Structure data will be reconstructed to match the convention of Quantum ESPRESSO.
- `nat` (the number of atoms) and `ntyp` (the number of element types) will be superseded by the values obtained from the crystal structure data.
- The cutoff values `ecutwfc` and `ecutrho` are obtained from the pseudo-potential files if these parameters are left blank.

CELL_PARAMETERS

- This block will not be generated if `use_ibrav` is set to `true`. Otherwise, the lattice vectors are exported in units of angstrom.
- The information of the lattice vectors are obtained from the crystal structure data. When the `data` field is defined and contains a 3x3 matrix, that value will be used for the set of lattice vectors instead.

ATOMIC_SPECIES

- This block exports a list of atom species, atomic mass, and the file name of the pseudo-potential data.
- The information of the atoms are obtained from the crystal structure data. The file names of the pseudo-potential data are referred from the CSV-formatted index file specified by `pp_list` parameter.
- When the `data` field is defined and contains the required data, these values will be used instead.

ATOMIC_POSITIONS

- This block exports the atomic species and their fractional coordinates.
- When `ignore_species` is given to specify an atomic species or a list of species, the values of `if_pos` for these species will be set to `0`. It is used for MD or structure relaxations.
- When the `data` field is defined and contains the required data, these values will be used instead.

K_POINTS

- This block exports the information of k points. The type of the output is specified by the `option` parameter that takes one of the following:
  - `gamma`: uses Γ point.
  - `crystal`: generates a list of k points in mesh pattern. The mesh width is given by the `grid` parameter, or derived from the `vol_density` or `k_resolution` parameters.
  - `automatic`: generates a mesh of k points. It is given by the `grid` parameter, or derived from the `vol_density` or `k_resolution` parameters. The shift is obtained from the `kshifts` parameter.
- The mesh width is determined in the following order:
  - the `grid` parameter, specified by a list of n_x, n_y, n_z, or a scalar value n. For the latter, $n_x = n_y = n_z = n$ is assumed.
  - derived from the `vol_density` parameter.
  - derived from the `k_resolution` parameter, whose default value is 0.15.
- When the `data` field is defined and contains the required data, these values will be used.

5.3 Parameters for VASP

The entries of `optional` section and `content` part of the `tasks` section specific to VASP are explained below.

5.3.1 optional

The type and the location of pseudo-potential files are specified.

According to pymatgen, the pseudo-potential files are obtained from `PMG_VASP_PSP_DIR`/*functional*/POTCAR.{element}(.gz) or `PMG_VASP_PSP_DIR`/*functional*/{element}/POTCAR, where `PMG_VASP_PSP_DIR` points to the directory and it is given in the configuration file `~/.config/.pmgrc.yaml` or by the environment variable of the same name. *functional* refers to the type of the pseudo-potential, whose value is predefined as `POT_GGA_PAW_PBE`, `POT_LDA_PAW`, etc.

`pseudo_functional`

This parameter specifies the type of the pseudo-potential. The relation to the *functional* value above is defined in the table of pymatgen, for example, by `PBE` to `POT_GGA_PAW_PBE`, or by `LDA` to `POT_LDA_PAW`, or in a similar manner.

When the `pseudo_dir` parameter is specified, it is used as the directory that holds the pseudo-potential files, ignoring the convention of pymatgen.

`psuedo_dir`

This parameter specifies the directory that holds the pseudo-potential files. The paths to the pseudo-potential file turn to `pseudo_dir`/POTCAR.{element}(.gz), or `pseudo_dir`/{element}/POTCAR.

5.3.2 tasks

The template files are assumed to be placed in the directory specified by the `template_dir` parameter by the names `INCAR`, `KPOINTS`, `POSCAR`, and `POTCAR`. The missing files will be ignored.

5.3.3 content

incar

- This block contains parameters described in the INCAR file

kpoints

- `type`

  The `type` parameter describes how KPOINTS are specified. The following values are allowed, with some types accepting parameters. See pymatgen.io.vasp manual for further details.

  - `automatic`

    parameter: `grid`

  - `gamma_automatic`

    parameter: `grid`, `shift`

  - `monkhorst_automatic`

    parameter: `grid`, `shift`

  - `automatic_density`

    parameter: `kppa`, `force_gamma`

  - `automatic_gamma_density`

    parameter: `grid_density`

  - `automatic_density_by_vol`

    parameter: `grid_density`, `force_gamma`

  - `automatic_density_by_lengths`

    parameter: `length_density`, `force_gamma`

  - `automatic_linemode`

    parameter: `division`, `path_type` (corresponding to the `path_type` parameter of HighSymmKpath.)

5.4 Parameters for OpenMX

The entries of `optional` section and `content` part of the `tasks` section specific to OpenMX are explained below.

5.4.1 optional

`data_path`

This parameter specifies the name of directory that holds files for pseudo-atomic orbitals and pseudo-potentials. It corresponds to the `DATA.PATH` parameter.

5.4.2 content

`precision`

This parameter specifies the set of pseudo-atomic orbitals listed in Tables 1 and 2 of Section 10.6 of the OpenMX manual. It is one of `quick`, `standard`, or `precise`. The default value is `quick`.

Chapter 6

Extension guide

6.1 Adding modes of Quantum ESPRESSO

In order to add supports to modes of Quantum ESPRESSO, the mapping between the modes and the transformation classes should be added to `create_modeproc()` function in `src/cif2x/qe/calc_mode.py`.

```
def create_modeproc(mode, qe):
    if mode in ["scf", "nscf"]:
        modeproc = QEmode_pw(qe)
    else:
        modeproc = QEmode_generic(qe)
    return modeproc
```

The transformation functionality for each mode is provided as a derived class of `QEmode_base` class. This class implements methods `update_namelist()` for updating the namelist block, and `update_cards()` for generating data of card blocks. In the current version, two classes are provided: `QEmode_pw` class for scf and nscf calculations of pw.x, and `QEmode_generic` class for generating output as-is.

```
class QEmode_base:
    def __init__(self, qe):
    def update_namelist(self, content):
    def update_cards(self, content):
```

For the namelist, the transformation class generates values for blank entries from crystal structure data and other sources. It may also force to set values such as the lattice parameters that are determined from the crystal structure data, or those that must be specified consistently with other parameters. The functions are provided for each mode separately.

For card blocks, a function is provided for each card, and the mapping between the card type and the function is given in the `card_table` variable. The method `update_cards()` in the base class picks up and runs the function associated to the card, and updates the content of the card. Of course, a new `update_cards()` function may be defined.

```
self.card_table = {
    'CELL_PARAMETERS': generate_cell_parameters,
    'ATOMIC_SPECIES': generate_atomic_species,
    'ATOMIC_POSITIONS': generate_atomic_positions,
    'K_POINTS': generate_k_points,
}
```

The functions for cards are gathered in `src/cif2x/qe/cards.py` with the function names as `generate_{card name}`. These functions takes parameters for card blocks as argument, and returns a dictionary containing the card name, the options, and the data field.