ALM Documentation

Release 2.0.0 beta

Terumasa Tadano

Jun 13, 2023

Contents:

CHAPTER 1

About

1.1 What is ALM?

ALM is an open source software designed for estimating harmonic and anharmonic force constants based on the supercell method.

1.2 Features

- Extraction of harmonic and anharmonic force constants based on the supercell approach
- Applicable to any crystal structures and low-dimensional systems
- Accurate treatment of translational and rotational invariance
- Interface to VASP, Quantum-ESPRESSO, xTAPP, and LAMMPS codes
- API for python and C++

1.3 Links

- Documentation : http://alm.readthedocs.io (this page)
- Git repository : http://github.com/ttadano/ALM

1.4 License

1.5 How to Cite ALM

Please cite the following article when you use ALM in :

> T. Tadano, XXX, XXX **XX**, XXXX (XXXX)

1.6 Acknowledgement

This project is/was partially supported by the following projects:

- Grant-in-Aid for Young Scientists (B) (16K17724)
- Grant-in-Aid for Scientific Research on Innovative Areas 'Materials Design through Computics: Complex Correlation and Non-Equilibrium Dynamics'. (http://computics-material.jp)

1.7 Author & Contact

International Center for Young Scientists (ICYS),
National Institute for Material Science (NIMS),
Japan

CHAPTER 2

Building ALM using conda

ALM is written in C++. To build it, a set of build tools is needed. Currently using conda gives a relatively simple and uniform way to perform building the ALM python module, ALM library for C++, and ALMM command executable. In this documentation, it is presented a step-by-step procedure to build them using conda.

2.1 Preparing build tools by conda

At first, it is recommended to prepare a conda environment by:

```
% conda create --name alm -c conda-forge python=3.8
```

Here the name of the conda environment is chosen `alm`. The detailed instruction about the conda environment is found here. To build ALM on linux or macOS, the following conda packages are installed by

```
% conda install -c conda-forge numpy scipy h5py compilers "libblas=*=*mkl" spglib␣
↪boost eigen cmake ipython mkl-include
```

2.2 Building ALM

Now the directory structure supposed in this document is shown as below:

```
$HOME
|-- alm
|   `-- ALM
|       |-- bin/
|       |-- include/
|       |-- lib/
|       |-- python/setup.py
|       |-- src/
|       |-- _build/
|       |-- CMakeLists.txt
|       |-- CMakeLists.txt.conda
|       `-- ...
|
|-- $CONDA_PREFIX/include
|-- $CONDA_PREFIX/include/eigen3
|-- $CONDA_PREFIX/lib
`-- ...
```

`alm` directory is created by us and we move to this directory. Now we are in `$HOME/alm`. In this direcotry, ALM is downloaded from github. `ALM` directorie is created running the following commands:

```
% git clone https://github.com/ttadano/ALM.git
```

Make sure that you are using develop branch by

```
% cd ALM
% git branch
* develop
```

When this is done on `$HOME/ALM`, the above directory structure is made. If git command doesn't exist in your system, it is also obtained from conda by `conda install git`.

2.2.1 Building ALM python module

The ALM python module is built on the directory `$HOME/alm/ALM/python`. So first you have to move to this directory. The build and installation in the user directory is done by

```
% python setup.py build
% pip install -e .
```

2.2.2 Building ALM executable and C++ library

If you need only ALM python module, this section can be skipped.

Let's assume we are in the directory `$HOME/alm/ALM` (see above *directory structure*). The ALM library for C++ is built using cmake. The cmake's configuration file has to have the filename `CMakeLists.txt`. So its example of `CMakeLists.txt.conda` is renamed to `CMakeLists.txt`, i.e.,

```
% cp CMakeLists.txt.conda CMakeLists.txt
```

Then this `CMakeLists.txt` may be modified appropriately when the following compilation fails. Using this `CMakeLists.txt`, the ALM library for c++ is built for Linux by

```
% mkdir _build && cd _build
% cmake ..
% make -j4
% make install
```

and for macOS

```
% mkdir _build && cd _build
% cmake -DCMAKE_C_COMPILER='clang' -DCMAKE_CXX_COMPILER='clang++' ..
% make -j4
% make install
```

To see detailed make log, `-DCMAKE_VERBOSE_MAKEFILE:BOOL=ON` option for `cmake` should be added.

The dynamic and static link libraries and the head file are installed at

- `$HOME/alm/ALM/lib/libalmcxx.dylib` or `$HOME/alm/ALM/lib/libalmcxx.so`
- `$HOME/alm/ALM/lib/libalmcxx.a`
- `$HOME/alm/ALM/include/alm.h`

The executable is found at

- `$HOME/alm/ALM/bin/alm`

To use the dynamic link library, it may be necessary to set `$LD_LIBRARY_PATH` to

```
% export LD_LIBRARY_PATH=$CONDA_PREFIX/lib:$HOME/alm/ALM/lib:$LD_LIBRARY_PATH
```

and to use the executable

```
% export LD_LIBRARY_PATH=$CONDA_PREFIX/lib:$LD_LIBRARY_PATH
```

CHAPTER 3

Building ALM using Makefile

ALM can also be built with the Makefile in the `src` subdirectory. This approach only generates the command line version of ALM (binary `alm`). Therefore, if you want to use ALM from python as well, please *build ALM using conda*.

3.1 Requirement

- C++ compiler (C++11 standard or newer)
- LAPACK
- Boost C++ library
- Eigen3 library
- spglib

3.2 How to install

1. Install the LAPACK, Boost C++, and Eigen3, and spglib.

   To install the Boost C++ library, please download a source file from the website and unpack the file. Then, copy the 'boost' subdirectory to the include folder in the home directory (or anywhere you like). This can be done as follows:

   ```
   $ cd
   $ mkdir etc; cd etc
   (Download a source file and mv it to ~/etc)
   $ tar xvf boost_x_yy_z.tar.bz2
   $ cd ../
   $ mkdir include; cd include
   $ ln -s ../etc/boost_x_yy_z/boost .
   ```

In this example, we made a symbolic link to the 'boost' subdirectory in `$HOME/include`. Instead of installing from source, you can install the Boost library with Homebrew on Mac.

In the same way, please install the Eigen3 include files as follows:

```
$ cd
$ mkdir etc; cd etc
(Download a source file and mv it to ~/etc)
$ tar xvf eigen-eigen-*.tar.bz2 (* is an array of letters and digits)
$ cd ../
$ cd include
$ ln -s ../etc/eigen-eigen-*/Eigen .
```

2. Clone ALM from the git repository and edit Makefile:

```
$ git clone https://github.com/ttadano/ALM
$ cd ALM/src/
(Edit Makefile.linux or Makefile.osx)
$ make -f Makefile.linux -j (or make -j Makefile.osx -j)
```

In the `src` directory, we provide sample Makefiles for linux (Intel compiler) and MacOS (GCC installed via homebrew) as shown below.

Listing 1: **Makefile.linux**

```
CXX = icpc
CXXFLAGS = -O2 -xHOST -qopenmp -std=c++11
INCLUDE = -I../include -I$(HOME)/include -I$(SPGLIB_ROOT)/include

CXXL = ${CXX}
LDFLAGS = -mkl -L$(SPGLIB_ROOT)/lib -lsymspg

LAPACK =
LIBS = ${LAPACK}
```

Listing 2: **Makefile.osx**

```
# Use gcc >= 4.8 to use OpenMP
# OpenMP-enabled gcc can be installed via homebrew
CXX = g++-9
CXXFLAGS = -O2 -fopenmp -std=c++11
INCLUDE = -I../include -I$(HOME)/include -I$(SPGLIB_ROOT)/include -I/usr/local/
↪include/eigen3/ -I/usr/local/include/

CXXL = ${CXX}
LDFLAGS = -lgomp -L$(SPGLIB_ROOT)/lib -lsymspg

LAPACK = -llapack -lblas
LIBS = ${LAPACK}
```

3. Modify `LD_LIBRARY_PATH` as follows:

```
bash, zsh
$ export LD_LIBRARY_PATH=$(HOME)/src/spglib/lib:$LD_LIBRARY_PATH

csh, tcsh
$ setenv LD_LIBRARY_PATH $(HOME)/src/spglib/lib:$LD_LIBRARY_PATH
```

CHAPTER 4

Making input file for command line

4.1 Format of input file

Each input file should consist of entry fields. Available entry fields are

&general, **&interaction**, **&cutoff**, **&cell**, **&position**, and **&optimize**.

Each entry field starts from the key label **&field** and ends at the terminate character "/". For example, &general entry field should be given like

```
&general
  # Comment line
  PREFIX = prefix
  MODE = fitting
/
```

Multiple entries can be put in a single line when separated by semicolon (';'). Also, characters put on the right of sharp ('#') will be neglected. Therefore, the above example is equivalent to the following:

```
&general
  PREFIX = prefix; MODE = fitting  # Comment line
/
```

Each variable should be written inside the appropriate entry field.

4.2 List of input variables

4.2.1 "&general"-field

- **PREFIX**-tag : Job prefix to be used for names of output files

  Default None

Type String

- **MODE**-tag = optimize | suggest

optimize	Estimate harmonic and anharmonic IFCs. This mode requires an appropriate &optimize field.
suggest	Suggests the displacement patterns necessary to estimate harmonic and anharmonic IFCS.

Default None

Type String

- **NAT**-tag : Number of atoms in the supercell

Default None

Type Integer

- **NKD**-tag : Number of atomic species

Default None

Type Integer

- **KD**-tag = Name[1], ... , Name[`NKD`]

Default None

Type Array of strings

Example In the case of GaAs with `NKD = 2`, it should be `KD = Ga As`.

- TOLERANCE-tag : Tolerance for finding symmetry operations

Default 1.0e-6

Type Double

- PRINTSYM-tag = 0 | 1

0	Symmetry operations won't be saved in "SYMM_INFO"
1	Symmetry operations will be saved in "SYMM_INFO"

Default 0

type Integer

- PERIODIC-tag = PERIODIC[1], PERIODIC[2], PERIODIC[3]

0	Do not consider periodic boundary conditions when searching for interacting atoms.
1	Consider periodic boundary conditions when searching for interacting atoms.

Default 1 1 1

type Array of integers

Description This tag is useful for generating interacting atoms in low dimensional systems. When `PERIODIC[i]` is zero, periodic boundary condition is turned off along the direction of the lattice vector $\boldsymbol{a}_i$.

4.2.2 "&interaction"-field

- **NORDER**-tag : The order of force constants to be calculated. Anharmonic terms up to $(m+1)$th order will be considered with `NORDER` = m.

Default None

Type Integer

Example `NORDER = 1` for calculate harmonic terms only, `NORDER = 2` to include cubic terms as well, and so on.

- NBODY-tag : Entry for excluding multiple-body clusters from anharmonic force constants

Default `NBODY` = [2, 3, 4, . . . , `NORDER` + 1]

Type Array of integers

Description This tag may be useful for excluding multi-body clusters which are supposedly less important. For example, a set of fourth-order IFCs $\{\Phi_{ijkl}\}$, where i, j, k, and l label atoms in the supercell, can be categorized into four different subsets; **on-site**, **two-body**, **three-body**, and **four-body** terms. Neglecting the Cartesian coordinates of IFCs for simplicity, each subset contains the IFC elements shown as follows:

on-site	$\{\Phi_{iiii}\}$
two-body	$\{\Phi_{iijj}\}$, $\{\Phi_{iiij}\}$ $(i \neq j)$
three-body	$\{\Phi_{iijk}\}$ $(i \neq j, i \neq k, j \neq k)$
four-body	$\{\Phi_{ijkl}\}$ (all subscripts are different from each other)

Since the four-body clusters are expected to be less important than the three-body and less-body clusters, you may want to exclude the four-body terms from the Taylor expansion potential because the number of such terms are huge. This can be done by setting the `NBODY` tag as `NBODY = 2 3 3` togather with `NORDER = 3`.

More examples `NORDER = 2; NBODY = 2 2` includes harmonic and cubic IFCs but excludes three-body clusters from the cubic terms.

`NORDER = 5; NBODY = 2 3 3 2 2` includes anharmonic terms up to the sixth-order, where the four-body clusters are excluded from the fourth-order IFCs, and the multi ($\geq$ 3)-body clusters are excluded from the fifth- and sixth-order IFCs.

4.2.3 "&cutoff"-field

In this entry field, one needs to specify cutoff radii of interaction for each order in units of Bohr. The cutoff radii should be defined for every possible pair of atomic elements. For example, the cutoff entry for a harmonic calculation (`NORDER = 1`) of Si (`NKD = 1`) may be like

```
&cutoff
 Si-Si 10.0
/
```

This means that the cutoff radii of 10 a_0 will be used for harmonic Si-Si terms. The first column should be element-name strings, which must be a member of the `KD`-tag, connected by a hyphen ('-').

When one wants to consider cubic terms (`NORDER = 2`), please specify the cutoff radius for the cubic terms in the third column as the following:

```
&cutoff
 Si-Si 10.0 5.6 # Pair r_{2} r_{3}
/
```

Instead of giving specific cutoff radii, one can write "None" as follows:

```
&cutoff
 Si-Si None 5.6
/
```

which means that all possible harmonic terms between Si-Si atoms will be included.

Note: Setting 'None' for anharmonic terms can greatly increase the number of parameters and thereby increase the computational cost.

When there are more than two atomic elements, please specify the cutoff radii between every possible pair of atomic elements. In the case of MgO (`NKD = 2`), the cutoff entry should be like

```
&cutoff
 Mg-Mg 8.0
 O-O 8.0
 Mg-O 10.0
/
```

which can equivalently be written by using the wild card ('*') as

```
&cutoff
 *-* 8.0
 Mg-O 10.0 # Overwrite the cutoff radius for Mg-O harmonic interactions
/
```

Important: Cutoff radii specified by an earlier entry will be overwritten by a new entry that comes later.

Once the cutoff radii are properly given, harmonic force constants $\Phi_{i,j}^{\mu,\nu}$ satisfying $r_{ij} \leq r_c^{\mathrm{KD}[i]-\mathrm{KD}[j]}$ will be searched.

In the case of cubic terms, force constants $\Phi_{ijk}^{\mu\nu\lambda}$ satisfying $r_{ij} \leq r_c^{\mathrm{KD}[i]-\mathrm{KD}[j]}$, $r_{ik} \leq r_c^{\mathrm{KD}[i]-\mathrm{KD}[k]}$, and $r_{jk} \leq r_c^{\mathrm{KD}[j]-\mathrm{KD}[k]}$ will be searched and determined by fitting.

4.2.4 "&cell"-field

Please give the cell parameters in this entry in units of Bohr as the following:

```
&cell
 a
 a11 a12 a13
 a21 a22 a23
 a31 a32 a33
/
```

The cell parameters are then given by $\vec{a}_1 = a \times (a_{11}, a_{12}, a_{13})$, $\vec{a}_2 = a \times (a_{21}, a_{22}, a_{23})$, and $\vec{a}_3 = a \times (a_{31}, a_{32}, a_{33})$.

4.2.5 "&position"-field

In this field, one needs to specify the atomic element and fractional coordinate of atoms in the supercell. Each line should be

```
ikd xf[1] xf[2] xf[3]
```

where *ikd* is an integer specifying the atomic element (*ikd* = 1, . . . , `NKD`) and *xf[i]* is the fractional coordinate of an atom. There should be `NAT` such lines in the &position entry field.

4.2.6 "&optimize"-field

This field is necessary when `MODE = optimize`.

- LMODEL-tag : Choise of the linear model used for estimating force constants

"least-squares", "LS", "OLS", 1	Ordinary least square
"elastic-net", "enet", 2	Elastic net

Default least-squares

Type String

Description When `LMODEL = ols`, the force constants are estimated from the displacement-force datasets via the ordinary least-squares (OLS), which is usually sufficient to calculate harmonic and third-order force constants.

The elestic net (`LMODEL = enet`) should be useful to calculate the fourth-order (and higher-order) force constants. When the elastic net is selected, the users have to set the following related tags: `CV`, `L1_RATIO`, `L1_ALPHA`, `CV_MAXALPHA`, `CV_MINALPHA`, `CV_NALPHA`, `STANDARDIZE`, `ENET_DNORM`, `MAXITER`, `CONV_TOL`, `NWRITE`, `SOLUTION_PATH`, `DEBIAS_OLS`

- **DFSET**-tag : File name containing displacement-force datasets for training

Default None

Type String

Description The format of `DFSET` can be found *here*

- NDATA-tag : Number of displacement-force training datasets

Default None

Type Integer

Description If `NDATA` is not given, the code reads all lines of `DFSET` (excluding comment lines) and estimates `NDATA` by dividing the line number by `NAT`. If the number of lines is not divisible by `NAT`, an error will be raised. `DFSET` should contain at least `NDATA`× `NAT` lines.

- NSTART, NEND-tags : Specifies the range of data to be used for training

Default `NSTART = 1`, `NEND = NDATA`

Type Integer

Example To use the data in the range of [20:30] out of 50 entries, the tags should be `NSTART = 20` and `NEND = 30`.

- SKIP-tag : Specifies the range of data to be skipped for training

  Default None

  Type Two integers connected by a hyphen

  Description `SKIP` =i-j skips the data in the range of [i:j]. The i and j must satisfy $1 \leq i \leq j \leq$ `NDATA`. This option may be useful when doing cross-validation manually (`CV=-1`).

- DFSET_CV-tag : File name containing displacement-force datasets used for manual cross-validation

  Default `DFSET_CV = DFSET`

  Type String

  Description This tag is used only when `LMODEL = enet` and `CV = -1`.

- NDATA_CV-tag : Number of displacement-force validation datasets

  Default None

  Type Integer

  Description This tag is used only when `LMODEL = enet` and `CV = -1`.

- NSTART_CV, NEND_CV-tags : Specifies the range of data to be used for validation

  Default `NSTART_CV = 1`, `NEND_CV = NDATA_CV`

  Type Integer

  Example This tag is used only when `LMODEL = enet` and `CV = -1`.

- CV-tag : Cross-validation mode for elastic net

0	Cross-validation mode is off. The elastic net optimization is solved with the given `L1_ALPHA` value. The force constants are written to `PREFIX`.fcs and `PREFIX`.xml.
> 0	`CV`-fold cross-validation is performed *automatically*. `NDATA` training datasets are divided into `CV` subsets, and `CV` different combinations of training-validation datasets are created internally. For each combination, the elastic net optimization is solved with the various `L1_ALPHA` values defined by the `CV_MINALPHA`, `CV_MAXALPHA`, and `CV_NALPHA` tags. The result of each cross-validation is stored in `PREFIX`.enet_cvset[1, . . . , `CV`], and their average and deviation are stored in `PREFIX`.cvscore.
-1	The cross-validation is performed *manually*. The Taylor expansion potential is trained by using the training datasets in `DFSET`, and the validation score is calculated by using the data in `DFSET_CV` for various `L1_ALPHA` values defined the `CV_MINALPHA`, `CV_MAXALPHA`, and `CV_NALPHA` tags. After the calculation, the fitting and validation errors are stored in `PREFIX`.enet_cv. This option may be convenient for a large-scale problem since multiple optimization tasks with different training-validation datasets can be done in parallel.

Default 0

Type Integer

- L1_ALPHA-tag : The coefficient of the L1 regularization term

  Default 0.0

  Type Double

  Description This tag is used only when `LMODEL = enet` and `CV = 0`.

- CV_MINALPHA, CV_MAXALPHA, CV_NALPHA-tags : Options to specify the `L1_ALPHA` values used in cross-validation

  Default `CV_MINALPHA = 1.0e-4`, `CV_MAXALPHA = 1.0`, `CV_NALPHA = 1`

  Type Double, Double, Integer

  Description `CV_NALPHA` values of `L1_ALPHA` are generated from `CV_MINALPHA` to `CV_MAXALPHA` in logarithmic scale. A recommended value of `CV_MAXALPHA` is printed out to the log file. This tag is used only when `LMODEL = enet` and the cross-validation mode is on (`CV > 0` or `CV = -1`).

- L1_RATIO-tag : The ratio of the L1 regularization term

  Default 1.0 (LASSO)

  Type Double

  Description The `L1_RATIO` changes the regularization term as `L1_ALPHA` $\times$ [`L1_RATIO` $|\boldsymbol{\Phi}|_1$ + $\frac{1}{2}$ (1-`L1_RATIO`) $|\boldsymbol{\Phi}|_2^2$]. Therefore, `L1_RATIO = 1` corresponds to LASSO. `L1_RATIO` must be `0 < L1_ratio <= 1`.

- STANDARDIZE-tag = 0 | 1

0	Do not standardize the sensing matrix
1	Each column of the sensing matrix is standardized in such a way that its mean value becomes 0 and standard deviation becomes 1.

  Default 1

  Type Integer

  Description This option influences the optimal `L1_ALPHA` value. So, if you change the `STANDARDIZE` option, you will have to rerun the cross-validation.

- ENET_DNORM-tag : Normalization factor of atomic displacements

  Default 1.0

  Type Double

  Description The normalization factor of atomic displacement u_0 in units of Bohr. When $u_0 (\neq 1)$ is given, the displacement data are scaled as $u_i \rightarrow u_i/u_0$ before constructing the sensing matrix. This option influences the optimal `L1_ALPHA` value. So, if you change the `ENET_DNORM` value, you will have to rerun the cross-validation. Also, this tag has no effect when `STANDARDIZE = 1`.

- MAXITER-tag : Number of maximum iterations of the coordinate descent algorithm

Default 10000

Type Integer

Description Effective when `LMODEL = enet`.

- CONV_TOL-tag : Convergence criterion of the coordinate descent iteration

Default 1.0e-8

Type Double

Description The coordinate descent iteration finishes at ith iteration if $\sqrt{\frac{1}{N}|\boldsymbol{\Phi}_i - \boldsymbol{\Phi}_{i-1}|_2^2} <$ `CONV_TOL` is satisfied, where N is the length of the vector $\boldsymbol{\Phi}$.

- SOLUTION_PATH-tag = 0 | 1

0	Do not save the solution path.
1	Save the solution path of each cross-validation combination in `PREFIX`.solution_path.

Default 0

Type Integer

Description Effective when `LMODEL = enet` and the cross-validation mode is on.

- DEBIAS_OLS-tag = 0 | 1

0	Save the solution of the elastic net problem to `PREFIX`.fcs and `PREFIX`.xml.
1	After the solution of the elastic net optimization problem is obtained, only non-zero coefficients are collected, and the ordinary least-squares fitting is solved again with the non-zero coefficients before saving the results to `PREFIX`.fcs and `PREFIX`.xml. This might be useful to reduce the bias of the elastic net solution.

Default 0

Type Integer

Description Effective when `LMODEL = enet` and `CV = 0`.

- ICONST-tag = 0 | 1 | 2 | 3 | 11

0	No constraints
1	Constraints for translational invariance will be imposed between IFCs. Available only when `LMODEL = ols`.
11	Same as `ICONST = 1` but the constraint is imposed *algebraically* rather than numerically. Select this option when `LMODEL = enet`.
2	In addition to `ICONST = 1`, constraints for rotational invariance will be imposed up to (`NORDER` + 1)th order. Available only when `LMODEL = ols`.
3	In addition to `ICONST = 2`, constraints for rotational invariance between (`NORDER` + 1)th order and (`NORDER` + 2)th order, which are zero, will be considered. Available only when `LMODEL = ols`.

Default 1

Type Integer

Description See *this page* for the numerical formulae.

- ROTAXIS-tag : Rotation axis used to estimate constraints for rotational invariance. This entry is necessary when `ICONST = 2, 3`.

  Default None

  Type String

  Example When one wants to consider the rotational invariance around the x-axis, one should give `ROTAXIS = x`. If one needs additional constraints for the rotation around the y-axis, `ROTAXIS` should be `ROTAXIS = xy`.

- FC2XML-tag : XML file to which the harmonic terms will be fixed upon fitting

  Default None

  Type String

  Description When `FC2XML`-tag is given, harmonic force constants will be fixed to the values stored in the `FC2XML` file. This may be useful for optimizing cubic and higher-order terms without changing the harmonic terms. Please make sure that the number of harmonic terms in the new computational condition is the same as that in the `FC2XML` file.

- FC3XML-tag : XML file to which the cubic terms will be fixed upon fitting

  Default None

  Type String

  Description Same as the `FC2XML`-tag, but `FC3XML` is to fix cubic force constants.

CHAPTER 5

Using ALM from python

ALM's python module can be made following *this page*.

The job of ALM is to select force constants elements among all possible elements for atoms in a supercell and then ALM fits relations between displacements and forces to those force constants elements. The selection of the force constants elements is done by the order of force constants, force constants symmetry, user-input cutoff distances, and maybe LASSO regression analysis. In the following, how to use ALM is presented step by step.

5.1 Initialization

ALM class instance is made by context manager as follows:

```python
from alm import ALM

with ALM(lavec, xcoord, numbers) as alm:
    ...
```

`lavec`, `xcoord`, and `numbers` are the essential parameters and are the basis vectors, fractional coordinates of atoms, and atomic numbers, respectively. `lavec` is the 3×3 matrix and a, b, c basis vectors are given as the row vectors, i.e.,

$$\begin{pmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{pmatrix}.$$

`xcoord` is the $n \times 3$ matrix. Each row gives the point coordinates of the atom. `numbers` is a list of integer numbers.

5.2 Dataset: displacements and forces

ALM requires a dataset composed of sets of pairs of atomic displacements from the input crystal structure and forces due to the displacements.

Because there should be many supercells with different configurations of displacements, the array shape of the displacement data is `(number of supercells, number of atoms, 3)`. The array shape of forces is the same as that of the displadements.

For example, the file format of datasets for ALM commandline interface (`DFSET`) can be easily tranformed to the displacements and forces by

```
dfset = np.loadtxt("DFSET").reshape((-1, number_of_atoms, 6))
displacements = dfset[:, :, :3]
forces = dfset[:, :, 3:]
```

These data are set to ALM by

```
from alm import ALM

with ALM(lavec, xcoord, numbers) as alm:
    ...
    alm.displacements = displacements
    alm.forces = forces
```

5.3 Selection of force constants elements

The basis force constants selection is performed by

```
alm.define(maxorder, cutoff_radii, nbody)
```

`maxorder = 1` for only harmonic (2nd-order) force constants, and `maxorder = 2` for 2nd- and 3rd-order force constants, and so on, i.e. up to (n+1)th order force constants are included in the consideration with `maxorder=n`. `cutoff_radii` controls how far the atomic pairs are searched to select force constants elements. `nbody` is used to limit interaction of atoms. The details meanings are found in *"&interaction"-field* for `maxorder` and `nbody` and in *"&cutoff"-field* for `cutoff_radii`.

5.4 Force constants calculation

Force constants are calculated by fitting dataset of displacements and forces to the force constants elements selected by

```
alm.optimize()
```

There are two solvers, which are chosen either by `solver='dense'` (default) or `solver='SimplicialLDLT'` as the keyword argument.

5.5 Extraction of force constants values

The calculated force constants elements are stored in ALM intance in the ALM's manner. They are extracted by

```
alm.get_fc(fc_order, mode)
```

By `fc_order=n`, (n+1)th order force constants are extracted. `mode` chooses the format of force constants. There are three modes, but two of them are important, which are `all` and `origin`. By `mode=all`, all elements of force constants are returned except for the elements whose values are 0. With `mode=origin`, the first atomic indices of force constants are limited for only those in the primitive cell.

5.6 LASSO and elastic net regression

As shown in *"&optimize"-field*, ALM has a functionality to compute force constants using LASSO and elestic net regression. This feature is accessed from ALM python module such as by

```
optcontrol = {'linear_model': 2,
              'cross_validation': 4,
              'num_l1_alpha': 50}
alm.optimizer_control = optcontrol
```

The controllable parameters and their variable types are listed as follows:

```
optimizer_control_data_types = OrderedDict([
    ('linear_model', int),                              # LMODEL
    ('use_sparse_solver', int),                         # '=1' equivalent to SimplicialLDLT
    ('maxnum_iteration', int),                          # MAXITER
    ('tolerance_iteration', float),                     # CONV_TOL
    ('output_frequency', int),                          # NWRITE
    ('standardize', int),                               # STANDARDIZE
    ('displacement_normalization_factor', float),       # ENET_DNORM
    ('debiase_after_l1opt', int),                       # DEBIAS_OLS
    ('cross_validation', int),                          # CV
    ('l1_alpha', float),                                # L1_ALPHA
    ('l1_alpha_min', float),                            # CV_MINALPHA
    ('l1_alpha_max', float),                            # CV_MAXALPHA
    ('num_l1_alpha', int),                              # CV_NALPHA
    ('l1_ratio', float),                                # L1_RATIO
    ('save_solution_path', int)])                       # SOLUTION_PATH
```

5.7 Wrap-up and example

Some examples are found in `example` directory.

The following python script is an example to computer 2nd and 3rd order force constants by ordinary least square fitting. Let's assume the crystal is wurtzite-type AlN as found in the example directory. For the 2nd order, no cutoff radii are used, but for the 3rd order, cutoff radii of 4 Angstrom is chosen between all pairs of atoms (Al-Al, Al-N, N-N). The fitting is achieved by using lapack SVD solver. Be sure that the memory usage is ~1.6GB and the whole calculation takes a few minutes, depending on computers though.

```
import h5py
import numpy as np
from alm import ALM

with open("POSCAR_AlN") as f:
    lines = f.readlines()
[lines.pop(0) for i in range(2)]
[lines.pop(3) for i in range(3)]
vals = np.array([np.fromstring(l, dtype='double', sep=' ') for l in lines])
lavec = vals[:3]
xcoord = vals[3:]
numbers = [13, ] * 36 + [7, ] * 36
natom = len(numbers)

dfset = np.loadtxt("DFSET_AlN").reshape((-1, natom, 6))
displacements = dfset[:, :, :3]
forces = dfset[:, :, 3:]

cutoff_radii = [np.ones((2, 2)) * -1, np.ones((2, 2)) * 4]

with ALM(lavec, xcoord, numbers, verbosity=1) as alm:
    alm.displacements = displacements
    alm.forces = forces
    alm.define(2, cutoff_radii=cutoff_radii)
    alm.optimize()

    fc2 = np.zeros((natom, natom, 3, 3), dtype='double', order='C')
    fc3 = np.zeros((natom, natom, natom, 3, 3, 3), dtype='double', order='C')
    for fc, indices in zip(*alm.get_fc(1, mode='all')):
        v1, v2 = indices // 3
        c1, c2 = indices % 3
        fc2[v1, v2, c1, c2] = fc
    for fc, indices in zip(*alm.get_fc(2, mode='all')):
        v1, v2, v3 = indices // 3
        c1, c2, c3 = indices % 3
        fc3[v1, v2, v3, c1, c2, c3] = fc
    with h5py.File('fc.hdf5', 'w') as w:
        w.create_dataset('fc2', data=fc2, compression='gzip')
        w.create_dataset('fc3', data=fc3, compression='gzip')
```

CHAPTER 6

How to make a DFSET file

6.1 Format of DFSET

The displacement-force data sets obtained by first-principles (or classical force-field) calculations have to be saved to a file, say *DFSET*. Then, the force constants are estimated by setting `DFSET` = *DFSET* and with `MODE = optimize`.

The *DFSET* file must contain the atomic displacements and corresponding forces in Cartesian coordinate for at least `NDATA` structures (displacement patterns) in the following format: Structure number 1 (this is just a comment line)

$$\begin{array}{ccc}
u_x(1) & u_y(1) & u_z(1) \quad \mathrm{f}_x(1) \\
f_y(1) & f_z(1) & \\
u_x(2) & u_y(2) & u_z(2) \quad \mathrm{f}_x(2) \\
f_y(2) & f_z(2) & \\
 & \vdots & \\
\vdots & & \\
u_x(\mathrm{NAT}) & u_y(\mathrm{NAT}) & u_z(\mathrm{NAT}) \quad \mathrm{f}_x(\mathrm{NAT}) \\
f_y(\mathrm{NAT}) & f_z(\mathrm{NAT}) &
\end{array}$$

Structure number 2

$$\begin{array}{ccc}
u_x(1) & u_y(1) & u_z(1) \quad \mathrm{f}_x(1) \\
f_y(1) & f_z(1) & \\
 & \vdots & \\
\vdots & &
\end{array}$$

Here, `NAT` is the number of atoms in the supercell. The unit of displacements and forces must be **Bohr** and **Ryd/Bohr**, respectively.

6.2 Generation of DFSET by extract.py

The script `extract.py` in the tools directory of ALM is useful to generate `DFSET` files from output files of some popular DFT codes. Let us assume that we have calculated atomic forces for 10 different structures by VASP and saved the results as vasprun_01.xml … vasprun_10.xml. Then, a `DFSET` file can be generated as:

VASP

```
$ python extract.py --VASP=SPOSCAR --offset=vasprun0.xml vasprun??.xml >␣
↪DFSET
```

Here, `SPOSCAR` is the supercell structure without atomic displacements, and vasprun0.xml is the result of DFT calculation for `SPOSCAR`. The `--offset` option subtract the offset (residual) components of atomic forces from the data in vasprun??.xml.

Important: The `--offset` is optional, but we strongly recommend to use it when the fractional coordinates of atoms have degrees of freedom.

The `extract.py` can also parse the data from the output files of QE, OpenMX, xTAPP, and LAMMPS:

QE

```
$ python extract.py --QE=supercell.pw.in --offset=supercell.pw.out disp??.pw.
↪out > DFSET
```

OpenMX

```
$ python extract.py --OpenMX=supercell.dat --offset=supercell.out disp??.out␣
↪> DFSET
```

xTAPP

```
$ python extract.py --xTAPP=supercell.cg --offset=supercell.str disp??.str >␣
↪DFSET
```

LAMMPS

The LAMMPS case requires a special treatment. We first need to add the *dump* option in the LAMMPS input file as

```
dump            1 all custom 1 XFSET id xu xy xz fx fy fz
dump_modify     1 format float "%20.15f"
```

This option will generate the file *XFSET* which contains atomic coordinates and forces. After generating *XFSET* files for 10 structures and save them as *XFSET.01* … *XFSET.10* , we can create `DFSET` as:

```
$ python extract.py --LAMMPS=supercell.lammps --offset=supercell.XFSET XFSET.
↪?? > DFSET
```

CHAPTER 7

Mathematical background

7.1 Interatomic force constants (IFCs)

The starting point of the computational methodology is to approximate the potential energy of interacting atoms by a Taylor expansion with respect to atomic displacements by

$$\begin{aligned} U - U_0 &= \sum_{n=1}^{N} U_n = U_1 + U_2 + U_3 + \cdots , \\ U_n &= \frac{1}{n!} \sum_{\substack{\ell_1\kappa_1,\dots,\ell_n\kappa_n \\ \mu_1,\dots,\mu_n}} \Phi_{\mu_1\dots\mu_n}(\ell_1\kappa_1;\dots;\ell_n\kappa_n)\, u_{\mu_1}(\ell_1\kappa_1)\cdots u_{\mu_n}(\ell_n\kappa_n). \end{aligned} \tag{7.1}$$

Here, $u_\mu(\ell\kappa)$ is the atomic displacement of κth atom in the ℓth unit cell along μth direction, and $\Phi_{\mu_1\dots\mu_n}(\ell_1\kappa_1;\dots;\ell_n\kappa_n)$ is the nth-order interatomic force constant (IFC).

7.2 Symmetry relationship between IFCs

The are several relationships between IFCs which may be used to reduce the number of independence IFCs.

- Permutation

  IFC should be invariant under the exchange of the triplet (ℓ, κ, μ), e.g.,

  $$\Phi_{\mu_1\mu_2\mu_3}(\ell_1\kappa_1;\ell_2\kappa_2;\ell_3\kappa_3) = \Phi_{\mu_1\mu_3\mu_2}(\ell_1\kappa_1;\ell_3\kappa_3;\ell_2\kappa_2) = \dots.$$

- Periodicity

  Since IFCs should depend on interatomic distances, they are invariant under a translation in units of the lattice vector, namely

  $$\Phi_{\mu_1\mu_2\dots\mu_n}(\ell_1\kappa_1;\ell_2\kappa_2;\dots;\ell_n\kappa_n) = \Phi_{\mu_1\mu_2\dots\mu_n}(0\kappa_1;\ell_2-\ell_1\kappa_2;\dots;\ell_n-\ell_1\kappa_n).$$

- Crystal symmetry

A crystal symmetry operation maps an atom $\vec{r}(\ell\kappa)$ to another equivalent atom $\vec{r}(LK)$ by rotation and translation. Since the potential energy is invariant under any crystal symmetry operations, IFCs should transform under a symmetry operation as follows:

$$\sum_{\nu_1,\dots,\nu_n} \Phi_{\nu_1\dots\nu_n}(L_1K_1;\dots;L_nK_n)O_{\nu_1\mu_1}\cdots O_{\nu_n\mu_n} = \Phi_{\mu_1\dots\mu_n}(\ell_1\kappa_1;\dots;\ell_n\kappa_n), \tag{7.2}$$

where O is the rotational matrix of the symmetry operation. Let N_s be the number of symmetry operations, there are N_s relationships between IFCs which may be used to find independent IFCs.

Note: In the implementation of ALM, symmetricaly independent IFCs are searched in Cartesian coordinate where the matrix element $O_{\mu\nu}$ is 0 or ± 1 in all symmetry operations except for those of **hexagonal** (trigonal) lattice. Also, except for hexagonal (trigonal) systems, the product $O_{\nu_1\mu_1}\cdots O_{\nu_n\mu_n}$ in the left hand side of equation (7.2) becomes non-zero only for a specific pair of $\{\nu\}$ (and becomes 0 for all other $\{\nu\}$s). Therefore, let $\{\nu'\}$ be such a pair of $\{\nu\}$, the equation (7.2) can be reduced to

$$\Phi_{\nu'_1\dots\nu'_n}(L_1K_1;\dots;L_nK_n) = s\Phi_{\mu_1\dots\mu_n}(\ell_1\kappa_1;\dots;\ell_n\kappa_n), \tag{7.3}$$

where $s = \pm 1$. The code employs equation (7.3) instead of equation (7.2) to reduce the number of IFCs. If IFCs of the left-hand side and the right-hand side of equation (7.3) are equivalent and the coupling coefficient is $s = -1$, the IFC is removed since it becomes zero. For **hexagonal** (trigonal) systems, there can be symmetry operations where multiple terms in the left-hand side of equation (7.2) become non-zero. For such cases, equation (7.2) is not used to reduce the number of IFCs. Alternatively, the corresponding symmetry relationships are imposed as constraints between IFCs in solving fitting problems.

7.3 Constraints between IFCs

Since the potential energy is invariant under rigid translation and rotation, it may be necessary for IFCs to satisfy corresponding constraints.

The constraints for translational invariance are given by

$$\sum_{\ell_1\kappa_1} \Phi_{\mu_1\mu_2\dots\mu_n}(\ell_1\kappa_1;\ell_2\kappa_2;\dots;\ell_n\kappa_n) = 0, \tag{7.4}$$

which should be satisfied for arbitrary pairs of $\ell_2\kappa_2,\dots,\ell_n\kappa_n$ and $\mu_1,\dots,\mu_n$. The code **alm** imposes equation (7.4) by default (`ICONST = 1`).

The constraints for rotational invariance are

$$\begin{aligned}\sum_{\ell'\kappa'}&(\Phi_{\mu_1\dots\mu_n\nu}(\ell_1\kappa_1;\dots;\ell_n\kappa_n;\ell'\kappa')r_\mu(\ell'\kappa') - \Phi_{\mu_1\dots\mu_n\mu}(\ell_1\kappa_1;\dots;\ell_n\kappa_n;\ell'\kappa')r_\nu(\ell'\kappa')) \\ &+ \sum_{\lambda=1}^{n}\sum_{\mu'_\lambda} \Phi_{\mu_1\dots\mu'_\lambda\dots\mu_n}(\ell_1\kappa_1;\dots;\ell_\lambda\kappa_\lambda;\dots;\ell_n\kappa_n)(\delta_{\mu,\mu_\lambda}\delta_{\nu,\mu'_\lambda} - \delta_{\nu,\mu_\lambda}\delta_{\mu,\mu'_\lambda}) = 0,\end{aligned}$$

which must be satisfied for arbitrary pairs of $(\ell_1\kappa_1,\dots,\ell_n\kappa_n;\mu_1,\dots,\mu_n;\mu,\nu)$. This is complicated since $(n+1)$th-order IFCs (first line) are related to nth-order IFCs (second line).

For example, the constraints for rotational invariance related to harmonic terms can be found as

$$\begin{aligned}\sum_{\ell_2\kappa_2}&(\Phi_{\mu_1\nu}(\ell_1\kappa_1;\ell_2\kappa_2)r_\mu(\ell_2\kappa_2) - \Phi_{\mu_1\mu}(\ell_1\kappa_1;\ell_2\kappa_2)r_\nu(\ell_2\kappa_2)) \\ &+ \Phi_\nu(\ell_1\kappa_1)\delta_{\mu,\mu_1} - \Phi_\mu(\ell_1\kappa_1)\delta_{\nu,\mu_1} = 0,\end{aligned}$$

and

$$\sum_{\ell_3\kappa_3}(\Phi_{\mu_1\mu_2\nu}(\ell_1\kappa_1;\ell_2\kappa_2;\ell_3\kappa_3)r_\mu(\ell_3\kappa_3) - \Phi_{\mu_1\mu_2\mu}(\ell_1\kappa_1;\ell_2\kappa_2;\ell_3\kappa_3)r_\nu(\ell_3\kappa_3))$$
$$+ \Phi_{\nu\mu_2}(\ell_1\kappa_1;\ell_2\kappa_2)\delta_{\mu,\mu_1} - \Phi_{\mu\mu_2}(\ell_1\kappa_1;\ell_2\kappa_2)\delta_{\nu,\mu_1}$$
$$+ \Phi_{\mu_1\nu}(\ell_1\kappa_1;\ell_2\kappa_2)\delta_{\mu,\mu_2} - \Phi_{\mu_1\mu}(\ell_1\kappa_1;\ell_2\kappa_2)\delta_{\nu,\mu_2} = 0.$$

When `NORDER = 1`, equation (7.5) will be considered if `ICONST = 2`, whereas equation (7.5) will be neglected. To further consider equation (7.5), please use `ICONST = 3`, though it may enforce a number of harmonic IFCs to be zero since cubic terms don't exist in harmonic calculations (`NORDER = 1`).

7.4 Estimate IFCs by linear regression

7.4.1 Basic notations

From the symmetrically independent set of IFCs and the constraints between them for satifying the translational and/or rotational invariance, we can construct an irreducible set of IFCs $\{\Phi_i\}$. Let us denote a column vector comprising the N irreducible set of IFCs as $\boldsymbol{\Phi}$. Then, the Taylor expansion potential (TEP) defined by equation (7.1) is written as

$$U_{\mathrm{TEP}} = \boldsymbol{b}^T\boldsymbol{\Phi}.$$

Here, $\boldsymbol{b} \in \mathbb{R}^{1\times N}$ is a function of atomic displacements $\{u_i\}$ defined as $\boldsymbol{b} = \partial U/\partial\boldsymbol{\Phi}$. The atomic forces based on the TEP is then given as

$$\boldsymbol{F}_{\mathrm{TEP}} = -\frac{\partial U_{\mathrm{TEP}}}{\partial \boldsymbol{u}} = -\frac{\partial \boldsymbol{b}^T}{\partial \boldsymbol{u}}\boldsymbol{\Phi} = A\boldsymbol{\Phi}, \tag{7.5}$$

where $A \in \mathbb{R}^{3N_s\times N}$ with N_s being the number of atoms in the supercell, and $\boldsymbol{u}^T = (u_1^x, u_1^y, u_1^z, \dots, u_{N_s}^x, u_{N_s}^y, u_{N_s}^z)$ is the vector comprising $3N_s$ atomic displacements in the supercell. Note that the matrix A and force vector $\boldsymbol{F}_{\mathrm{TEP}}$ depend on the atomic configuration of the supercell. To make this point clearer, let us denote them as $A(\boldsymbol{u})$ and $\boldsymbol{F}_{\mathrm{TEP}}(\boldsymbol{u})$.

To estimate the IFC vector $\boldsymbol{\Phi}$ by linear regression, it is usually necessary to consider several different displacement patterns. Let us suppose we have N_d displacement patterns and atomic forces for each pattern obtained by DFT. Then, equation (7.5) defined for each displacement pattern can be combined to single equation as

$$\boldsymbol{F}_{\mathrm{TEP}} = \mathbb{A}\boldsymbol{\Phi},$$

where $\boldsymbol{F}^T = [\boldsymbol{F}^T(\boldsymbol{u}_1), \dots, \boldsymbol{F}^T(\boldsymbol{u}_{N_d})]$ and $\mathbb{A}^T = [A^T(\boldsymbol{u}_1), \dots, A^T(\boldsymbol{u}_{N_d})]$.

7.4.2 Ordinary least-squares

In the ordinary least-squares (`LMODEL = least-squares`), IFCs are estimated by solving the following problem:

$$\boldsymbol{\Phi}_{\mathrm{OLS}} = \underset{\boldsymbol{\Phi}}{\mathrm{argmin}}\,\frac{1}{2N_d}\|\boldsymbol{F}_{\mathrm{DFT}} - \boldsymbol{F}_{\mathrm{TEP}}\|_2^2 = \underset{\boldsymbol{\Phi}}{\mathrm{argmin}}\,\frac{1}{2N_d}\|\boldsymbol{F}_{\mathrm{DFT}} - \mathbb{A}\boldsymbol{\Phi}\|_2^2. \tag{7.6}$$

Therefore, the IFCs are determined so that the residual sum of squares (RSS) is minimized. To determine all elements of $\boldsymbol{\Phi}_{\mathrm{OLS}}$ uniquely, $\mathbb{A}^T\mathbb{A}$ must be full rank. When the fitting is successful, **alm** reports the relative fitting error σ defined by

$$\sigma = \sqrt{\frac{\|\boldsymbol{F}_{\mathrm{DFT}} - \mathbb{A}\boldsymbol{\Phi}\|_2^2}{\|\boldsymbol{F}_{\mathrm{DFT}}\|_2^2}}, \tag{7.7}$$

where the denominator is the square sum of the DFT forces.

7.4.3 Elastic-net regression

In the elasitc-net optimization (`LMODEL = elastic-net`), IFCs are estimated by solving the following optimization problem:

$$\boldsymbol{\Phi}_{\mathrm{enet}} = \underset{\boldsymbol{\Phi}}{\operatorname{argmin}} \frac{1}{2N_d} \| \boldsymbol{F}_{\mathrm{DFT}} - \mathbb{A}\boldsymbol{\Phi} \|_2^2 + \alpha\beta \|\boldsymbol{\Phi}\|_1 + \frac{1}{2}\alpha(1-\beta)\|\boldsymbol{\Phi}\|_2^2, \tag{7.8}$$

where α is a hyperparameter that controls the trade-off between the sparsity and accuracy of the model, and β $(0 < \beta \leq 1)$ is a hyperparameter that controls the ratio of the L_1 and L_2 regularization terms. α and β must be given by input tags `L1_ALPHA` and `L1_RATIO`, respectively.

An optimal value of α can be estimated, for example, by cross-validation (CV). A n-fold CV can be performed by setting the `CV`-tag properly.

CHAPTER 8

API

8.1 ALM python modules

8.1.1 alm module

8.1.2 Module contents

CHAPTER 9

Indices and tables

- genindex
- modindex
- search