ablCS Documentation

Release 2.1.0

ablCS's team

Jun 12, 2023

CONTENTS:

CHAPTER

ONE

ABOUT ABICS

1.1 What is abICS ?

abICS is a software framework for training a machine learning model to reproduce first-principles energies and then using the model to perform configurational sampling in disordered systems. Specific emphasis is placed on multi-component solid state systems such as metal and oxide alloys. The current version of abics can use neural network models implemented in aenet to be used as the machine learning model. As of this moment, abICS can also generate Quantum Espresso, VASP, and OpenMX input files for obtaining the reference training data for the machine learning model. For the sampling algorithms, abICS implements the extended Monte Carlo methods, namely, the replica exchange Monte Carlo method (RXMC) and the population annealing Monte Carlo method (PAMC). In addition, as a beta version, the grand canonical sampling is supported.

1.2 Developers

abICS is developed by the following members.

- **ver. 2.0**
  - Shusuke Kasamatsu (Yamagata University)
  - Yuichi Motoyama (Institute for Solid State Physics, Univ. of Tokyo)
  - Kazuyoshi Yoshimi (Institute for Solid State Physics, Univ. of Tokyo)
  - Tatsumi Aoyama (Institute for Solid State Physics, Univ. of Tokyo)
  - Osamu Sugino (Institute for Solid State Physics, Univ. of Tokyo)
- **ver. 1.0**
  - Shusuke Kasamatsu (Yamagata University)
  - Yuichi Motoyama (Institute for Solid State Physics, Univ. of Tokyo)
  - Kazuyoshi Yoshimi (Institute for Solid State Physics, Univ. of Tokyo)
  - Yoshiyuki Yamamoto (Institute for Solid State Physics, Univ. of Tokyo)
  - Osamu Sugino (Institute for Solid State Physics, Univ. of Tokyo)
  - Taisuke Ozaki (Institute for Solid State Physics, Univ. of Tokyo)

1.3 Version information

- ver. 2.1.0 : 2023/06/12.
- ver. 2.0.1 : 2022/11/04.
- ver. 2.0 : 2022/06/24.
- ver. 1.0 : 2020/05/01.
- ver. 1.0-beta : 2020/03/31.
- ver. 0.1 : 2019/12/10.

1.4 License

1.5 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.

CHAPTER

TWO

INSTALL

2.1 Prerequisites

abICS requires Python3 (>=3.7).

The following Python packages are required.

- numpy
- scipy
- toml
- mpi4py
- pymatgen (>=2019.12.3)
- qe-tools

These are installed automatically but mpi4py and pymatgen need extra software-packages before installing them.

- mpi4py needs one of the MPI implementations, e.g., Open MPI.
- pymatgen needs Cython:

```
$ pip3 install cython
```

2.2 Install from PyPI

Since abICS is registered in PyPI users can install abICS easily:

```
$ pip3 install abics
```

If you want to install abICS locally because, for example, you have no permission to write files, the following command:

```
$ pip3 install --user abics
```

installs abICS below a directory `~/.local` . If you want to install abICS into another directory, use the `--prefix=DIRECTORY` option (`DIRECTORY` is the path to the directory where abICS will be installed) .

2.3 Install from source

2.3.1 Download

The source codes of abICS can be obtained from GitHub page .

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

2.3.2 Directory structure

The directory structure of abICS is given as follows:

```
.
|-- COPYING
|-- README.md
|-- abics/
|   |-- __init__.py
|   |-- applications/
|   |-- exception.py
|   |-- mc.py
|   |-- mc_mpi.py
|   |-- replica_params.py
|   |-- scripts/
|   |-- util.py
|-- docs/
|   |-- sphinx/
|-- examples/
|-- pyproject.toml
|-- test/
|-- tests/
```

A set of python modules are located in the `abics` directory.

2.3.3 Install

- Pass the location of the root directory of abICS as an argument of `pip3 install` :

  $ pip3 install ./abICS

2.4 Uninstall

- `pip3 uninstall abics` uninstalls abics from your machine.

CHAPTER

THREE

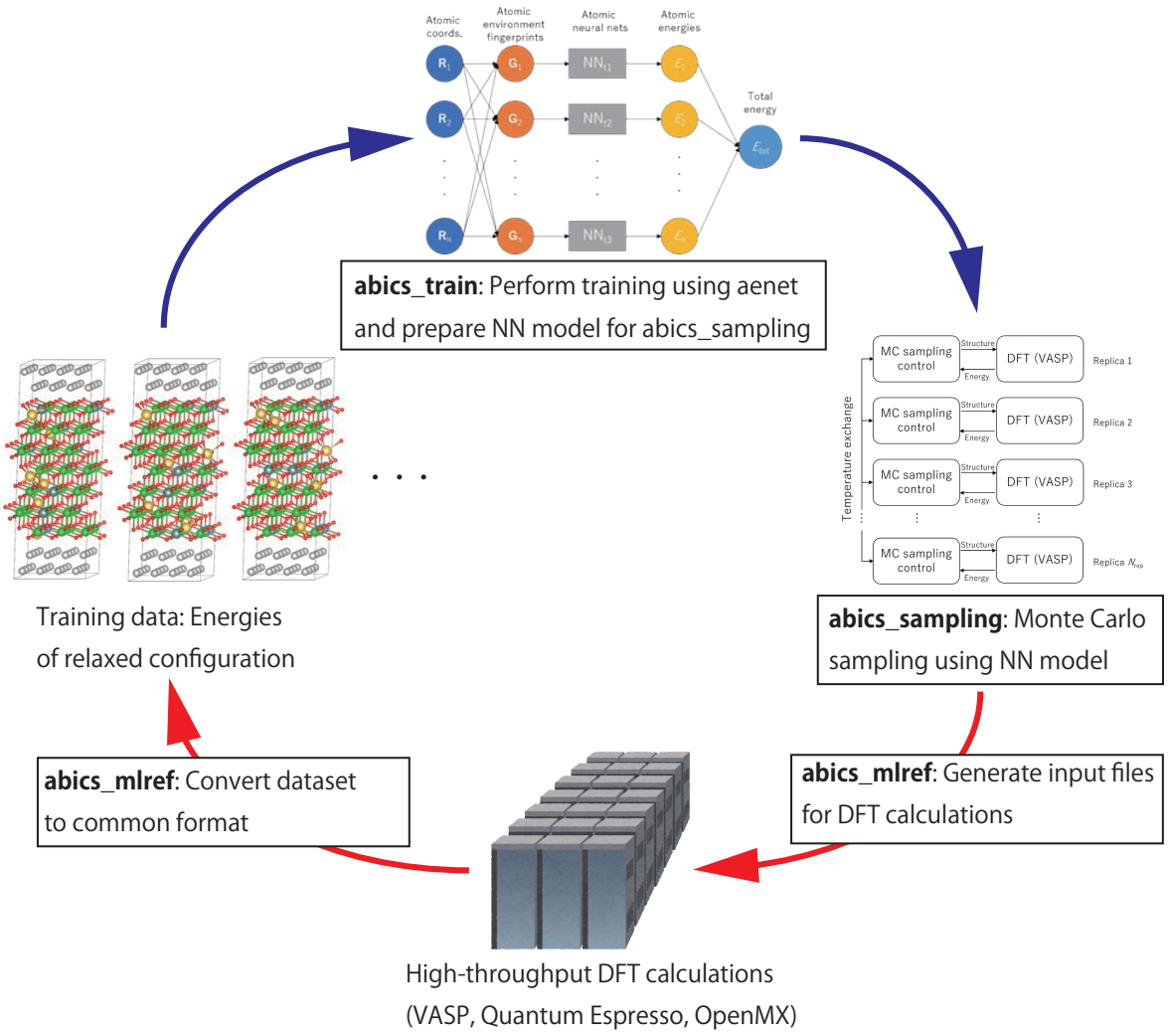
BASIC USAGE

3.1 Active learning

abICS was originally developed for directly combining first-principles calculations with replica-exchange Monte Carlo methods to perform configurational sampling, but the scale of the models and the number of steps that can be calculated are limited by the large computational cost of first-principles calculations. In contrast, Ver. 2 implements an active learning method to construct a neural network model that can rapidly predict the energy after structural optimization, dramatically improving the sampling speed [Kasamatsu et al., 2022] .

The general flow of the active learning method implemented in abICS is as follows.

1. Perform ab initio calculations on a large number of randomly generated atomic configurations and prepare training data (correspondence between configurations and energies).
2. Build a neural network model that predicts energy from atomic configurations using the prepared training data.
3. Perform statistical thermodynamic sampling of atomic configurations using a replica exchange Monte Carlo method with a neural network model.
4. Evaluate the accuracy of the neural network model by sampling the ion configurations that appear in the Monte Carlo calculations and performing ab initio calculations on each of them.
5. If the accuracy is not sufficient, add the results calculated in 4. to the training data and repeat from 2.



Schematic of the active learning procedure using abICS

3.2 Preparing an abICS control file

First, we have to prepare an input file that controls the entire abICS framework. The input file of abICS is comprised of the following five sections:

1. [sampling] section specifies the parameters of the replica exchange Monte Carlo part, such as the number of replicas, the temperature range, and the number of Monte Carlo steps. In addition, [sampling.solver] subsection specifies the parameters for the (first principle calculation) solver, including the type of solver (VASP, QE,...), the path to the solver, and the directory containing immutable input files.
2. [mlref] section specifies options for extracting only atomic configurations from the sampling results in order to evaluate the accuracy of the neural network model and to expand the training data. In addition, for generating training data, [mlref.solver] subsection specifies the parameters for the (first principle calculation) solver, including the type of solver (VASP, QE,...), the path to the solver, and the directory containing immutable input files. This section is used for `abics_mlref` .
3. [train] section specifies optinons for making a trainer to learn a placement energy prediction model from training data. This section is used for `abics_train` .

4. [observer] section specifies the type of physical quantity to be calculated.
5. [config] section specifies the configuration of the alloy, etc.

For details, see *Input Files Format* . The following is an example of an input file selecting aenet as a solver.

```
[sampling]
nreplicas = 8
nprocs_per_replica = 1
kTstart = 600.0
kTend = 2000.0
nsteps = 6400 # Number of steps for sampling
RXtrial_frequency = 4
sample_frequency = 16
print_frequency = 1
reload = false

[sampling.solver]
type = 'aenet'
path= 'predict.x-2.0.4-ifort_serial'
base_input_dir = './baseinput'
perturb = 0.0
run_scheme = 'subprocess' #'mpi_spawn_ready'
ignore_species = ["O"]

[mlref]
nreplicas = 8
ndata = 5

[mlref.solver]
type = 'qe'
base_input_dir = './baseinput_ref'
perturb = 0.05
ignore_species = []

[train]
type = 'aenet'
base_input_dir = './aenet_train_input'
exe_command = ['generate.x-2.0.4-ifort_serial', 'srun train.x-2.0.4-ifort_intelmpi']
ignore_species = ["O"]
vac_map = []
restart = false

[config]
unitcell = [[8.1135997772, 0.0000000000, 0.0000000000],
            [0.0000000000, 8.1135997772, 0.0000000000],
            [0.0000000000, 0.0000000000, 8.1135997772]]
supercell = [1,1,1]

[[config.base_structure]]
type = "O"
coords = [
     [0.237399980, 0.237399980, 0.237399980],
     [0.762599945, 0.762599945, 0.762599945],
```

(continues on next page)

(continued from previous page)

```
    [0.512599945, 0.012600004, 0.737399936],
    [0.487399966, 0.987399936, 0.262599975],
    [0.012600004, 0.737399936, 0.512599945],
    [0.987399936, 0.262599975, 0.487399966],
    [0.737399936, 0.512599945, 0.012600004],
    [0.262599975, 0.487399966, 0.987399936],
    [0.987399936, 0.487399966, 0.262599975],
    [0.012600004, 0.512599945, 0.737399936],
    [0.487399966, 0.262599975, 0.987399936],
    [0.512599945, 0.737399936, 0.012600004],
    [0.262599975, 0.987399936, 0.487399966],
    [0.737399936, 0.012600004, 0.512599945],
    [0.237399980, 0.737399936, 0.737399936],
    [0.762599945, 0.262599975, 0.262599975],
    [0.512599945, 0.512599945, 0.237399980],
    [0.487399966, 0.487399966, 0.762599945],
    [0.012600004, 0.237399980, 0.012600004],
    [0.987399936, 0.762599945, 0.987399936],
    [0.987399936, 0.987399936, 0.762599945],
    [0.012600004, 0.012600004, 0.237399980],
    [0.487399966, 0.762599945, 0.487399966],
    [0.512599945, 0.237399980, 0.512599945],
    [0.737399936, 0.237399980, 0.737399936],
    [0.262599975, 0.762599945, 0.262599975],
    [0.237399980, 0.512599945, 0.512599945],
    [0.762599945, 0.487399966, 0.487399966],
    [0.762599945, 0.987399936, 0.987399936],
    [0.237399980, 0.012600004, 0.012600004],
    [0.737399936, 0.737399936, 0.237399980],
    [0.262599975, 0.262599975, 0.762599945],
    ]

[[config.defect_structure]]
coords = [
    [0.000000000, 0.000000000, 0.000000000],
    [0.749999940, 0.249999985, 0.499999970],
    [0.249999985, 0.749999940, 0.499999970],
    [0.249999985, 0.499999970, 0.749999940],
    [0.749999940, 0.499999970, 0.249999985],
    [0.499999970, 0.749999940, 0.249999985],
    [0.499999970, 0.249999985, 0.749999940],
    [0.000000000, 0.499999970, 0.499999970],
    [0.749999940, 0.749999940, 0.000000000],
    [0.249999985, 0.249999985, 0.000000000],
    [0.249999985, 0.000000000, 0.249999985],
    [0.749999940, 0.000000000, 0.749999940],
    [0.499999970, 0.000000000, 0.499999970],
    [0.000000000, 0.749999940, 0.749999940],
    [0.000000000, 0.249999985, 0.249999985],
    [0.499999970, 0.499999970, 0.000000000],
    [0.374999970, 0.374999970, 0.374999970],
    [0.624999940, 0.624999940, 0.624999940],
```

(continues on next page)

(continued from previous page)

```
     [0.374999970, 0.874999940, 0.874999940],
     [0.624999940, 0.124999993, 0.124999993],
     [0.874999940, 0.874999940, 0.374999970],
     [0.124999993, 0.124999993, 0.624999940],
     [0.874999940, 0.374999970, 0.874999940],
     [0.124999993, 0.624999940, 0.124999993],
     ]
[[config.defect_structure.groups]]
name = 'Al'
# species = ['Al']    # default
# coords = [[[0,0,0]]]  # default
num = 16 #432 #16000
[[config.defect_structure.groups]]
name = 'Mg'
# species = ['Mg']    # default
# coords = [[[0,0,0]]]  # default
num = 8 #216 #8000


[observer]
reference_structure = "MgAl2O4.vasp"
ignored_species = ["Al", "O"]
```

3.3 Preparing a reference file for first-principles solvers

The user must prepare reference input file(s) for first-principles solvers that are used for generating the training data according to the input format of the solver. The path of the reference file is specified by `base_input_dir` in the `[mlref.solver]` section in the abICS input file (see below). The coordinate information should not be written here; it will be written by abICS. The following is an example of a QE reference file.

```
&CONTROL
  calculation = 'relax'
  tstress = .false.
  tprnfor = .false.
  pseudo_dir = './pseudo'
  disk_io = 'low'
  wf_collect = .false.
/
&SYSTEM
  ecutwfc      =  60.0
  occupations  = "smearing"
  smearing     = "gauss"
  degauss      = 0.01
/
&electrons
  mixing_beta = 0.7
  conv_thr = 1.0d-8
  electron_maxstep = 100
/
&ions
```

(continues on next page)

(continued from previous page)

```
/
ATOMIC_SPECIES
Al 26.981 Al.pbe-nl-kjpaw_psl.1.0.0.UPF
Mg 24.305 Mg.pbe-spnl-kjpaw_psl.1.0.0.UPF
O  16.000 O.pbe-n-kjpaw_psl.1.0.0.UPF
ATOMIC_POSITIONS crystal

K_POINTS gamma
```

3.3.1 Specific notes for first-principles solvers

Every setting other than those pertaining to the atomic coordinates should be set in the solver reference input file. However, the specification of atoms to be relaxed can be controlled by abICS. To enable the structural optimization option, please set the option to do structural optimization in the reference file of the solver, and also specify the related parameters such as the total number of steps for relaxation. Furthermore, to interoperate with abICS, there are a few rules regarding the file name and contents of the reference file for each solver. We'll explain them below.

VASP

- URL : https://www.vasp.at
- Reference file rules
  - Please prepare INCAR, POTCAR, KPOINTS files.
    - In POTCAR file, please arrange the atoms in alphabetical order.
    - The POSCAR file is basically not needed, but maybe needed depending on the version of `pymatgen`. In that case, please prepare a suitable file.

Quantum Espresso

- URL : https://www.quantum-espresso.org
- Available version: 6.2 or higher
  - "Old XML" format is not available
- Reference file rules
  - Please set the reference file name as `scf.in` .
  - `calculation` option must be `scf` or `relax` .
  - If the calculation is done only at Γ point, the calculation becomes fast if you set `Gamma` in `kpoints`.

OpenMX

- URL : http://www.openmx-square.org
- Available version: 3.9
- Reference file rule
  - Please set the reference file name as `base.dat` .

3.4 Preparing a reference file for training and evaluating the machine learning model

The user must prepare a reference file according to the input format of the machine learning model trainer and calculator to be used (only aenet is supported at the moment). The path of the reference file is specified by `base_input_dir` in the `[solver]` section in the abICS input file (see below). The coordinate information should not be written here because it will obviously change in the course of the simulation. The lattice sites are specified in a separate abICS input file (see below), and abICS will take care of generating the coordinates section at each sampling step.

3.4.1 Machine learning trainer/calculator-specific notes

aenet

- URL : http://ann.atomistic.net
- Checked with version 2.0.4.
- Reference file rules
  - Place the input files for aenet in the `generate`, `train`, and `predict` directories in the directory which is set in the `base_input_dir` of the `[trainer]` section.
  - aenet compiles the atomic configuration and energy data for training into an intermediate binary format that is converted to atomic environment descriptor-energy relationships before training. Please place the input files for this conversion `generate.x` in the `generate` directory.
  - Place an input file for `train.x` in the `train` directory that reads the training data generated by `generate.x` and trains. The file should be named `train.in`.
  - Place the input file `predict.in` for `predict.x` in the `predict` directory to evaluate the energy for the input coordinates using the trained potential model.
- abICS control file
  - In the `[solver]` section, for `type` , `perturb` , and `run_scheme`, set the following if using an active learning scheme.

```
type = "aenet"
perturb = 0.0
run_scheme = 'subprocess'
```

3.5 Creating a set of training data

1. Generate a set of input files for the first-principle calculation using `abics_mlref`.
2. Perform the first-principle calculation with these inputs. (In the tutorial GNU parallel is used for the high-throughput calculation.)

3.6 Creating a neural network

1. Run `abics_mlref` again to convert the results of the first-principle calculation into a common format that `abics_train` will read.
2. Execute `abics_train` to create a neural network. When the calculation is completed successfully, the trained neural network is output in `baseinput` directory.

3.7 Monte Carlo sampling

By using `abics_sampling`, Monte Carlo sampling can be performed by using the trained neural network. (The number of MPI processes must be larger than the number of replicas.) Running the program will create directories named by the replica numbers under the current directory, and each replica runs the solver in it.

abICS can call the `aenet` library via the LAMMPS interface (`aenetPyLammps`). This is faster than calling `aenet` directly because it does not need file I/O. To use `aenetPyLammps`, you need to install `aenet-lammps` and `lammps`. For details, please refer to the *Predict energy of annet model via LAMMPS interface*.

CHAPTER

FOUR

TUTORIAL

In this tutorial, we demonstrate the calculation of the degree of inversion of Mg and Al atoms in an ionic crystal $MgAl_2O_4$. Input files are provided in `examples/active_learning_qe/` .

4.1 Constructing a neural network model

This section contains instructions on how to construct a neural network using `aenet` to reproduce first-principles energies from Quantum ESPRESSO (QE). Here, we will consider the temperature-dependent degree of Mg/Al inversion in MgAl2O4 spinel. In the ground state, all Mg ions are tetrahedrally coordinated by O ions while Al ions are octahedrally coordinated. We will simulate the amount of inversion between these sites vs. temperature. The set of input files used in this tutorial can be found in `examples/active_learning_qe`. In the following, we briefly describe the installation of aenet and GNU parallel, but you may skip the section if they are preinstalled in your system. We will also use ohtaka, the supercomputer system B of the Institute for Solid State Physics, as the environment for running the calculations.

4.1.1 Preparation

Installation of aenet

In abICS, we use aenet to build neural network models. You can download aenet from http://ann.atomistic.net. Follow the Installation instructions in the Documentation to install it. Note that abICS uses `generate.x`, `train.x` and `predict.x` of aenet for training and evaluating neural networks. For `train.x`, an MPI parallel version can be used, but for `generate.x` and `predict.x`, you need to use a non-MPI executable file (serial). For this reason, you should also install the serial version under makefiles.

Installation of GNU parallel

In this tutorial, we will use GNU parallel to run first-principles calculations with Quantum Espresso in parallel. Therefore, you need to install GNU parallel first. GNU parallel can be downloaded from https://www.gnu.org/software/parallel/ (on a Mac, it can also be installed directly by homebrew). After moving to the directory where you downloaded and extracted the source files, you can install it into `$HOME/opt/parallel` by typing the following command.

```
$ ./configure --prefix=$HOME/opt/parallel
$ make && make install
```

For detailed configuration, please refer to the official manual.

4.1.2 Preparation of input files for training data set generation

A set of training data is required for creating a neural network that relate the configurations of atoms as input and the energy as output by the first-principle calculations. To generate the data set, the input files need to be prepared for both abICS and the first-principle solver.

Preparation of the abICS control file (`input.toml`)

This file contains the definition of the lattice structure to be calculated, the control of the entire active learning cycles by abICS, and the parameters for the replica exchange Monte Carlo method. By using the st2abics tool, you can automatically generate the input.toml template from the crystal structure file.

```
$ cd [example_dir].
$ st2abics st2abics_MgAl2O4.toml MgAl2O4.vasp > input.toml
```

In this section, we will explain the settings for each section of `input.toml` in more detail.

(i) `[mlref]` section

```
[mlref]
nreplicas = 8
ndata = 5
```

In this section, you can set the options for extracting atomic configurations from the RXMC calculation results to evaluate the accuracy of the neural network model and to expand the training data. Basically, `nreplicas` should be the same values as in the `[sampling]` section. `ndata` specifies how many samples to be extracted as the training dataset of the machine learning model from configurations generated by the RXMC calculation. Therefore, it should be set to a value less than or equal to the number of configurations generated by the RXMC calculation, `nsteps/sample_frequency` in `[sampling]` section.

(ii) `[mlref.solver]` section

```
[mlref.solver] # Set up a reference ab initio solver.
type = 'qe'
base_input_dir = ['./baseinput_ref', './baseinput_ref', './baseinput_ref'] #, './
↪baseinput_ref']
perturb = 0.05
```

Set up the solver used to calculate the energy for training data (configuration energy). In this example, Quantum Espresso is used. The `base_input_dir` can be set freely. The input files for the solver are placed in the set directory (see below). If multiple directories are set up in a list format, as in this example, calculations using each input are performed in turn. The second and subsequent calculations will use the structure from the last step of the previous calculation as the initial coordinates. The energy of the last calculation is then used for training of the neural network model. For example, one could perform a fast structural optimization in the first input file at the expense of accuracy, and then perform a structural optimization in the second and subsequent input files with a higher accuracy setting. For another example, in the case of a lattice vector relaxation, the same input can be run multiple times to reset the computational mesh based on a set plane-wave cutoff.

The `perturb` is for starting the structural optimization from a structure with broken symmetry by randomly displacing each atom. In this case, the first calculation starts from the structure in which all atoms for structural relaxation are displaced by 0.05 angstrom in a random direction.

The `ignore-species` is set to an empty list when the first-principle solver is used for generating the training data. When a model is employed for the data generation in which some atomic species are ignored, they are specified in `ignore-species`.

(iii) `[config]` section

```
[config] # Set up information about the crystal lattice and the atoms and vacancies on␣
↪the lattice.
unitcell = [[8.1135997772, 0.0000000000000000, 0.0000000000000000],
            [0.0000000000000000, 8.1135997772, 0.0000000000000000],
            [0.0000000000000000, 0.0000000000000000, 8.1135997772]]
supercell = [1,1,1]

[[config.base_structure]]
type = "O"
coords = [
    [0.237399980, 0.237399980, 0.237399980],
    [0.762599945, 0.762599945, 0.762599945],
    [0.512599945, 0.012600004, 0.737399936],
    [0.487399966, 0.987399936, 0.262599975],
    ...
```

`[config]` section specifies atomic positions to be used in the Monte Carlo sampling. The `st2abics` utility tool can generate this section. If `abics_sampling` has not been performed yet, the atomic positions are randomly generated based on this information, and the input files for the first-principle calculation are produced. Once `abics_sampling` is executed, the input files will be generated from the atomic positions obtained from the Monte Carlo sampling.

Preparation of the QE reference file

Place the input file to be referenced in the QE scf calculation in `baseinput_ref`. The following is a description of the `scf.in` file in the sample directory.

```
&CONTROL
calculation = 'relax'
tstress = .false.
tprnfor = .false.
pseudo_dir = './pseudo'
disk_io = 'low'
wf_collect = .false.
/
&SYSTEM
  ecutwfc = 60.0
  occupations = "smearing".
  smearing = "gauss"
  degauss = 0.01
/
&electrons
  mixing_beta = 0.7
  conv_thr = 1.0d-8
  electron_maxstep = 100
/
```

(continues on next page)

(continued from previous page)

```
&ions
/
ATOMIC_SPECIES
Al 26.981 Al.pbe-nl-kjpaw_psl.1.0.0.UPF
Mg 24.305 Mg.pbe-spnl-kjpaw_psl.1.0.0.UPF
O 16.000 O.pbe-n-kjpaw_psl.1.0.0.UPF
ATOMIC_POSITIONS crystal

K_POINTS gamma
```

You need to rewrite the directory that contains the pseudopotentials, `pseudo_dir`, and the pseudopotentials used in `ATOMIC_SPECIES` according to your environment. The pseudopotentials used in this sample can be downloaded from the following link.

- https://pseudopotentials.quantum-espresso.org/upf_files/Al.pbe-nl-kjpaw_psl.1.0.0.UPF
- https://pseudopotentials.quantum-espresso.org/upf_files/Mg.pbe-spnl-kjpaw_psl.1.0.0.UPF
- https://pseudopotentials.quantum-espresso.org/upf_files/O.pbe-n-kjpaw_psl.1.0.0.UPF

In this example, `calculation = 'relax'` is used for structural optimization during the QE calculation, and `gammma` is used for `K_POINTS` to speed up the calculation.

4.1.3 Preparation of input files for training the neural network

In this tutorial we use `aenet` to train the neural netowrk. We need to prepare the input files for `abICS` and `aenet`.

Preparation of the abICS control file (`input.toml`)

(i) `[train]` section

```
[train] # Configure the model trainer.
type = 'aenet'
base_input_dir = '. /aenet_train_input'
exe_command = ['generate.x-2.0.4-ifort_serial',
               'srun train.x-2.0.4-ifort_intelmpi']
ignore_species = ["O"]
```

Set up a trainer to train a configuration energy prediction model from training data. Currently, abICS supports only aenet. You can freely set the `base_input_dir`. In that directory, set up the configuration files for the trainer (see below). In `exe_command`, specify the paths to `generate.x` and `train.x` of aenet. For `train.x`, an MPI parallel version is available, in which case, set the commands for MPI execution (`mpiexec`, `srun`, etc.) as shown in the example above.

The `ignore-species` is set to an empty list when the first-principle solver is used for generating the training data. When a model is employed for the data generation in which some atomic species are ignored, they are specified in `ignore-species`.

Preparation of input files for aenet

Place the input files for aenet in the `generate`, `train`, and `predict` directories in the directory set in the `base_input_dir` of the `[train]` section.

generate

aenet compiles the atomic configuration and energy data for training into an intermediate binary format that is converted into atomic environment descriptor-energy relationships before training. Input files for `generate.x` that perform this conversion are placed in the `generate` directory.

First, prepare a descriptor setting file for each element type. The file names are arbitrary. In the tutorial we will use `Al.fingerprint.stp`, `Mg.fingerprint.stp` and so on.

As an example, the content of `Al.fingerprint.stp` is shown below:

```
DESCR
 N. Artrith and A. Urban, Comput. Mater. Sci. 114 (2016) 135-150.
 N. Artrith, A. Urban, and G. Ceder, Phys. Rev. B 96 (2017) 014112.
END DESCR

ATOM Al # Specify element

ENV 2 # Specify the number of element species and element names that interact with the␣
↪element specified in ATOM
Al
Mg

RMIN 0.55d0 # Nearest neighbor distance between atoms

BASIS type=Chebyshev # Chebyshev Descriptor Settings
radial_Rc = 8.0  radial_N = 16 angular_Rc = 6.5  angular_N = 4
```

Please refer to the aenet documentation for more information on descriptor settings.

Next, prepare a file named `generate.in.head` as follows

```
OUTPUT aenet.train

TYPES
2
Al -0.0  ! eV
Mg -0.0  ! eV

SETUPS
Al   Al.fingerprint.stp
Mg    Mg.fingerprint.stp
```

`OUTPUT` must be set to `aenet.train`. Under `TYPES` specify the elemental species in the train data and their number. You can also specify an energy criterion for each elemental species, but it is basically safe to set it to 0. Under `SETUPS` specify the descriptor setup file for each elemental species. Be sure to include a newline at the end of the file. abICS will add a list of coordinate files to the end of `generate.in.head`, generate `generate.in`, and run `generate.x`.

train

Place the input file for `train.x`, which reads the training data generated by `generate` and trains, in the `train` directory. The file name should be `train.in`.

```
TRAININGSET aenet.train
TESTPERCENT 10
ITERATIONS  500

MAXENERGY 10000

TIMING

!SAVE_ENERGIES

METHOD
bfgs

NETWORKS
! atom   network         hidden
! types  file-name       layers  nodes:activation
  Al     Al.15t-15t.nn    2      15:tanh 15:tanh
  Mg       Mg.15t-15t.nn    2      15:tanh 15:tanh
```

Basically, no changes are needed except for the `NETWORKS` section. The `NETWORKS` section specifies the name of the potential file for each element species to be generated, the neural network structure, and the activation function.

predict

Place the input file `predict.in` for `predict.x` in the `predict` directory to evaluate the energy for the input coordinates using the trained potential model.

```
TYPES
2
Mg
Al

NETWORKS
Mg  Mg.15t-15t.nn
Al  Al.15t-15t.nn

VERBOSITY low
```

Enter the number of elemental species and their names in the `TYPES` section and the name of the potential file (set in `train.in`) for each elemental species in the `NETWORKS` section.

Also, `VERBOSITY` must be set to `low`.

Running the calculation

Now the input files have been prepared, we proceed to describe how to run the calculation. A sample script `AL.sh` is prepared to simplify the calculation procedure. `run_pw.sh` is used to run QE calculations; it is called inside `parallel_run.sh`, which will be described later. The contents of `AL.sh` is as follows.

```
#!/bin/sh
#SBATCH -p i8cpu
#SBATCH -N 4
#SBATCH -n 512
#SBATCH -J spinel
#SBATCH -c 1
#SBATCH --time=0:30:00

# Run reference DFT calc.
echo start AL sample
srun -n 8 abics_mlref input.toml >> abics_mlref.out

echo start parallel_run 1
sh parallel_run.sh

echo start AL final
srun -n 8 abics_mlref input.toml >> abics_mlref.out

#train
echo start training
abics_train input.toml >> abics_train.out

echo Done
```

The lines starting with `#SBATCH` and `srun` command are parameters of the job scheduler and the command to invoke parallel program (similar to `mpiexec`) used on the ISSP supercomputer system B, respectively. In this example, we are running an MPI parallel with 512 processes. For more information about the job scheduler, please refer to the manuals of your machine.

```
# Run reference DFT calc.
echo start AL sample
srun -n 8 abics_mlref input.toml >> abics_mlref.out
```

The above code block generates an input file for ab initio calculation, which is the main source of the training data, using `abics_mlref`. At the first execution, the specified number of atomic arrangements are randomly generated, a separate directory is prepared for each atomic arrangement, and an input file is created in the directory. At the same time, a file `rundirs.txt` is generated with the path of those directories. This directory listing can be used to automate the execution of ab initio computation jobs for individual inputs. We will then run the ab initio calculation based on the resulting file.

```
echo start parallel_run 1
sh parallel_run.sh
```

`parallel_run.sh` is a script to run high-throughput QE calculations in parallel using gnu parallel. It will manage the parallel running of calculations for the directories listed in rundirs.txt. The results of the QE calculation will be stored in each directory. Now that we have created the training data by the QE coverage calculation, we will move on to create the neural network potential in aenet. First, we run `abics_mlref` again to create files with the results of the ab initio calculations in a common format that abics_train will read.

```
echo start AL final
srun -n 8 abics_mlref input.toml >> abics_mlref.out
```

Next, we use anet to create a neural network potential based on the training data. The neural network potential is calculated by `abics_train`. The calculation is performed by reading the input file stored in `base_input_dir` in the `[train]` section of the input file. When the calculation is completed successfully, the trained neural network is output to the baseinput directory.

```
#train
echo start training
abics_train input.toml >> abics_train.out
```

The above process completes the AL.sh process for active learning.

4.2 Monte Carlo sampling

Next, we use the trained neural network potential for Monte Carlo samplings by abICS.

4.2.1 Preparation of input files

Several parameters need to be set in the abICS control file to perform the sampling as follows.

Preparation of the abICS control file (`input.toml`)

The calculation parameters are specified in `[sampling]` section concerning the Replica Exchange Monte carlo method.

(i) `[sampling]` section

```
[sampling]
nreplicas = 8
nprocs_per_replica = 1
kTstart = 600.0
kTend = 2000.0
nsteps = 6400
RXtrial_frequency = 4
sample_frequency = 16
print_frequency = 1
reload = false
```

In this section, you can configure settings related to the number of replicas, temperature range, etc. for the Replica Exchange Monte Carlo (RXMC) method (manual reference link). This time, we will use anet's `predict.x` as the energy solver for RXMC calculations. Currently, the mpi version of `predict.x` is not supported, so nprocs_per_replica should be 1.

(ii) [sampling.solver] section

```
[sampling.solver] # Configure the solver used for RXMC calculations
type = 'aenet'
path= 'predict.x-2.0.4-ifort_serial'
base_input_dir = '. /baseinput'
perturb = 0.0
run_scheme = 'subprocess'
ignore_species = ["O"]
```

In this section, you can configure the energy calculator (solver) to be used for RXMC calculations. In this tutorial, we will use `aenet` package to evaluate the neural network model. For `type`, `perturb`, and `run_scheme`, if you are using the active learning scheme, do not change the above example. Set path to the path of aenet's `predict.x` in your environment. The `base_input_dir`, where the input files corresponding to `predict.x` are generated, can be set freely (explained in detail later).

You can also specify the atomic species to be ignored in the neural network model as `ignore_species`. In this example, the sublattice of oxygen always has an occupancy of 1, so oxygens do not affect energy. In this case, it is more computationally efficient to ignore the existence when training and evaluating the neural network model.

4.2.2 Running the calculation

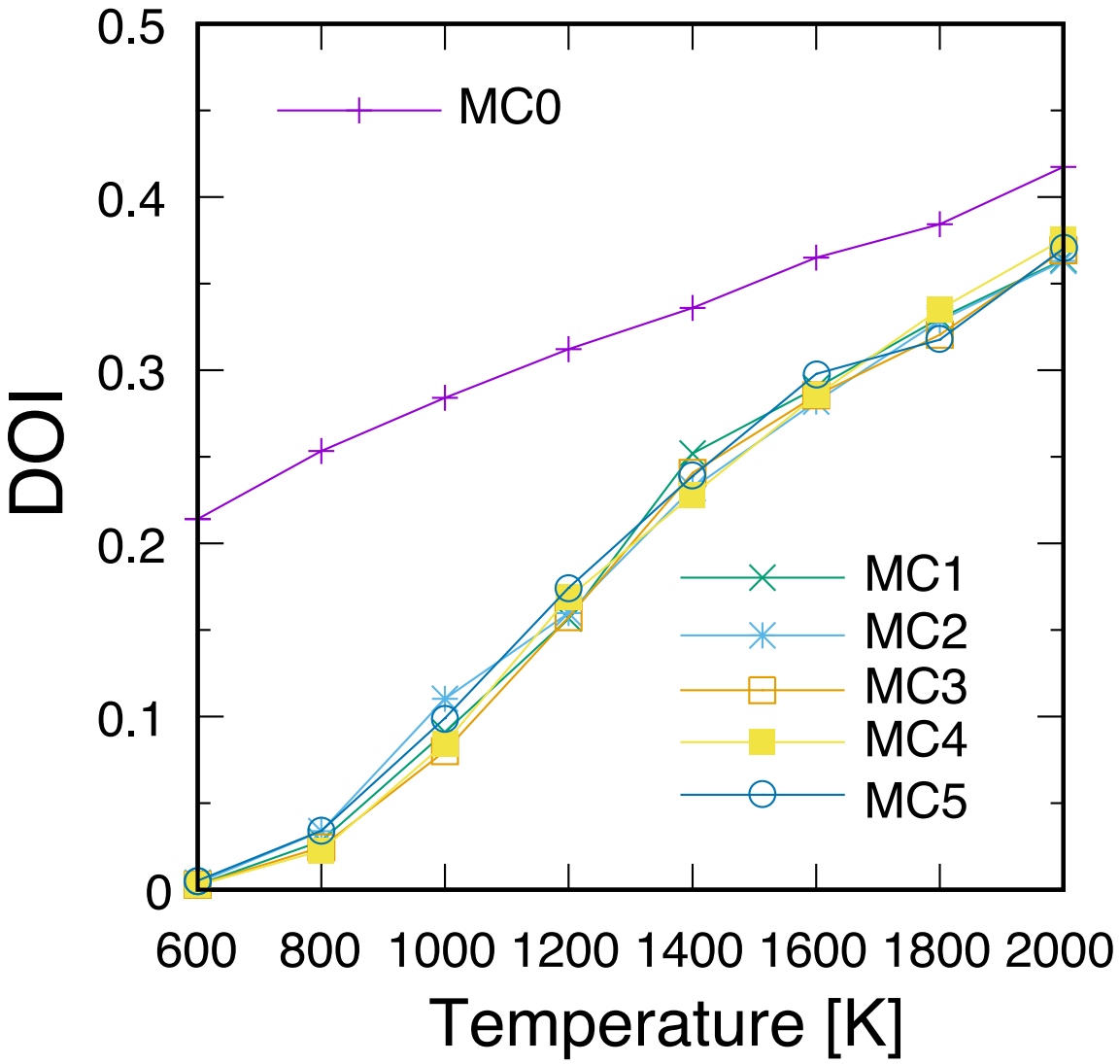
The sample script `MC.sh` is provided to simplify the calculation procedure. The content of the script is as follows.

```
#! /bin/sh
#SBATCH -p i8cpu
#SBATCH -N 1
#SBATCH -n 8
#SBATCH --time=00:30:00

srun -n 8 abics_sampling input.toml >> abics_sampling.out

echo Done
```

Running abicsAL will create the `MCxx` directory (where xx is the number of runs). With active learning in mind, additional functions have been implemented to obtain information such as the number of calculations by reading `ALloop.progress`. Under the `MCxx` directory, a folder will be created for the number of replicas. Then, in these folders, the atomic arrangement (`structure.XXX.vasp`) for each step described in the VASP POSCAR file format, the atomic position given the lowest energy (`minE.vasp`), and each step temperature and energy (`obs.dat`) etc. are output. For more details, please refer to the abICS manual output file.

The results obtained by the above procedure depend on the accuracy of the neural network potential computed by aenet. In the first step, we trained based on random configurations, thus the accuracy for low temperature structures is expected to be low. Here, by repeating the step of calculating the energy again by first-principles calculation for the structure estimated by Monte Carlo and relearning it, we expect to improve the accuracy in the whole temperature range.

This process can be calculated by repeating `AL.sh` and `MC.sh` in turn. The actual result of the calculation of the inversion rate (DOI) is shown in the figure below. In this example, the first result is `MC0`, followed by `MC1`, `MC2`, and so on. The first run is quite different from the others, thus we can expect that it is not accurate. On the other hand, if we train on the results of one Monte Carlo run, we find that the values are almost identical from the next run.



The DOI can be calculated by the following procedure.

1. Go to `MCxxx` directory.
2. Create `Tseparate` directory by `srun -n 8 abicsRXsepT ../input.toml`. (The number of processes should be the same as the number of replicas for `abics_sampling`. In this tutorial, the number of parallelism is set to 8, so set it to 8.)
3. copy `calc_DOI.py` and `MgAl2O4.vasp` in the sample directory.
4. Calculate the degree of inversion for each temperature by `srun -n 8 python3 calc_DOI.py ../input.toml`. (Align with the number of parallelism when `abics_sampling` is executed. In this tutorial, the number of parallelism is set to 8, so set it to 8.)

In general, you will need to write your own scripts (`calc_DOI.py` in the current example) for calculating thermody-

namic averages from the structures accumulated for each temperature in `MCxxx/Tseparate`.

Also, please note that the number of Monte Carlo steps in this example input is not enough for fully converging the degree of inversion. It is recommended to perform a separate RXMC calculation using the obtained neural network model with a larger number of sampling steps to calculate thermodynamic averages.

4.2.3 Predict energy of annet model via LAMMPS interface

abICS can call the `aenet` library via the LAMMPS interface (`aenetPyLammps`). This is faster than calling `aenet` directly because it does not need file I/O. The set of input files used in this tutorial can be found in `examples/active_learning_qe`.

Install aenetPyLammps

To use `aenetPyLammps`, you need to install `aenet-lammps` and `lammps`.

- URL : https://github.com/HidekiMori-CIT/aenet-lammps
- Use the commit 5d0f4bca .
  - `git checkout 5d0f4bca`
- Please install `aenet-lammps` according to the procedure specified in the above URL. Below are notes on installation.
  - `aenet`
    - Make sure to add `-fPIC` to `FCFLAGS` in `makefiles/Makefile.*`.
  - `lammps`
    - Make sure to add `LMP_INC = -DLAMMPS_EXCEPTIONS` in `src/Makefile`.
    - Make sure to add `mode=shared` to the make command option as `make mode=shared mpi` (when GCC, for example).
  - After completing the above installation, run `make install-python`.
    - `lammps` python package will be installed to the Python environment which is invoked by `python` command.

Training

The training procedure is the same as the previous section.

Sampling

Input file for prediction

Instead of the input file `predict.in` for `predict.x`, place the input file `in.lammps` under the `predict`:

```
pair_style      aenet
pair_coeff      * * v00 Al Mg 15t-15t.nn Al Mg
neighbor        0.1 bin
```

The detailed format of `in.lammmps` is written in the README of `aenet-lammps` repository.

Input for Sampling

Change the `type` and `run_scheme` in the `[sampling.solver]` section of the input file to `'aenetPyLammps'` and `'function'`, respectively.

```
[sampling.solver]
type = 'aenetPyLammps'
base_input_dir = ['./baseinput']
perturb = 0.0
run_scheme = 'function'
```

CHAPTER

FIVE

INPUT FILES FORMAT

The input file of abICS is constructed by the following five sections:

1. [sampling] section specifies the parameters of the replica exchange Monte Carlo part, such as the number of replicas, the temperature range, and the number of Monte Carlo steps. In addition, [sampling.solver] subsection specifies the parameters for the (first principle calculation) solver, including the type of solver (VASP, QE,...), the path to the solver, and the directory containing immutable input files.
2. [mlref] section specifies options for extracting only atomic configurations from the sampling results in order to evaluate the accuracy of the neural network model and to expand the training data. In addition, for generating training data, [mlref.solver] subsection specifies the parameters for the (first principle calculation) solver, including the type of solver (VASP, QE,...), the path to the solver, and the directory containing immutable input files. This section is used for `abics_mlref` .
3. [train] section specifies optinons for making a trainer to learn a placement energy prediction model from training data. This section is used for `abics_train` .
4. [observer] section specifies the type of physical quantity to be calculated.
5. [config] section specifies the configuration of the alloy, etc.
6. [log] section specifies the settings of logging.

The following sections describe the detail of each section.

5.1 [sampling] section

Specify the parameters of the Monte Carlo (MC) sampling method, such as the number of replicas, the temperature range, and the number of Monte Carlo steps. The example is shown as follows.

```
[sampling]
nreplicas = 3
nprocs_per_replica = 1
kTstart = 500.0
kTend = 1500.0
nsteps = 5
RXtrial_frequency = 2
sample_frequency = 1
print_frequency = 1
```

5.1.1 Input Format

Specify a keyword and its value in the form `keyword = value`. Comments can also be entered by adding # (Subsequent characters are ignored).

5.1.2 Keywords

- About sampling method
  - `sampler`

    Format : string

    Description : Relica exchange MC method ("RXMC") or population annealing MC method ("PAMC").
- About temperatures
  - Specify temperature points by using `kTs` or `kTstart`, `kTend`, and `kTnum` (lineary spaced). If `kTs` is specified, the others will be ignored.
  - `kTs`

    Format : list of float (>0)

    Description : Temperature points. When `sampler = "RXMC"`, the number of temperature points should equal to `nreplicas`.
  - `kTstart`

    Format : float (>0)

    Description : Minimum temperature.
  - `kTend`

    Format : float (>0)

    Description : Maximum temperature.
  - `kTnum` (Only for PAMC)

    Format : int (>0)

    Description : The number of temperature points. When `sampler = "RXMC"`, the number of temperature points will equal to `nreplicas`.
- About replica
  - `nprocs_per_replica`

    Format : int (natural number)

    Description : The number of processes for the replica. Default value = 1.
  - `nreplicas`

    Format : int (natural number)

    Description : The number of replicas.
- Others
  - `nsteps`

Format : int (natural number)

Description : Number of Monte Carlo steps.

- `nsteps_between_annealing` (Only for `sampler = "PAMC"`)

  Format : int (natural number)

  Description : Number of Monte Carlo steps for each temperature.

- `RXtrial_frequency` (Only for `sampler = "RXMC"`)

  Format : int (natural number)

  Description : The interval for performing replica exchange trials. For example, setting this value to 1 means that replica exchange is attempted at every Monte Carlo step, while setting this to 2 means that exchange is attempted at every second step. Default = 1.

- `resample_frequency` (Only for `sampler = "PAMC"`)

  Format : int (natural number)

  Description : The interval for performing replica resampling. For example, setting this value to 1 means that replica resampling is attempted after every temperature lowering, while setting this to 2 means that resampling is attempted at every second step. Default = 1.

- `sample_frequency`

  Format : int (natural number)

  Description : The interval for observation of physical quantities. Default value = 1.

- `print_frequency`

  Format : int (natural number)

  Description : The interval for saving physical quantities. Default value = 1.

- `reload`

  Format : bool ("true" or "false")

  Description : Whether to restart a prior calculation from the last step finished last time. Default value = false.

- `throw_out`

  Format : int or float

  Description : The number (int) or ratio (float) of measurements to be thrown out as thermalization in the process of the evaluation of expectation values. Default value = 0.5 .

- `enable_grandcanonical`

  Format : bool ("true" or "false")

  Description : Whether to allow grand canonical sampling. Default value = false.

- `gc_ratio`

  Format : float

  Description : The ratio of the grand canonical update that changes the number of elements among the trials of configuration updates, when the grand canonical sampling is turned on. Default value = 0.3 .

5.2 [mlref] section

Set options for retrieving only atomic configurations from the results of RXMC calculations. This is used, for example, to evaluate the accuracy of neural network models and to extend the training data. The file format is as follows.

```
[mlref]
nreplicas = 3
ndata = 50
```

5.2.1 Input Format

Keywords and their values are specified by a keyword and its value in the form `keyword = value`. Comments can also be entered by adding # (Subsequent characters are ignored).

5.2.2 Key words

- About replica
  - `nreplicas`

    Format : int (natural number)

    Description : The number of replicas.
  - `ndata`

    Format : int (natural number)

    Description : The number of data (configuration) to be sampled
  - `sampler`

    Format : string (default: "linspace")

    Description : The method to extract N_{data} samples from N samples generated by Monte Carlo method.
    - "linspace"

      Extract equilispaced samples by `numpy.linspace(0, N-1, num=ndata, dtype=int)`
    - "random"

      Random sampling by `numpy.random.choice(range(N), size=ndata, replace=False)`

5.3 [sampling.solver], [mlref.solver] section

These sections specify the parameters of solvers, which calculate the energy of a configuration (e.g., atomic positions). `sampling.solver` is used while Monte Carlo sampling, and `mlref.solver` is used for generating training dataset of a machine learning model.

In the present version, there are two main types of solvers.

- Ab initio solvers
  - Solvers for calculating DFT energy from atomic position.
  - Indeed, abICS uses an external DFT solver package such as VASP.

    * Parameters of the DFT solver such as convergence criteria are specified by using input files of the solver.
    * Solver specific notes (e.g., input filename) are described in *Specific notes for first-principles solvers*.
  - Configurations (atomic positions, atomic species, etc.) are specified by the `[config]` section.
- Potts model solver
  - A solver for calculating the energy $E = -\sum_{ij} \delta_{\sigma_i,\sigma_j} \quad (\sigma_i = 0, 1, \ldots, Q-1)$ of a spin configuration on a hyper cubic lattice ($\{\sigma_i\}$, $\sigma_i = 0, 1, \ldots, Q-1$).
  - The dimension and lengths of a hyper cubic lattice L, and the local degree of freedom of a spin Q are specified in the `[config]` section.
  - This solver is for the purpose of testing of algorithms and tutorials.

This section specifies solver parameters such as solver type (VASP, QE, ...), path to solver, directory with solver-specific input file(s). An example is shown as follows:

```
[sampling.solver]
type = 'vasp'
path = './vasp'
base_input_dir = './baseinput'
perturb = 0.1
run_scheme = 'mpi_spawn_ready'
```

5.3.1 Input Format

Keywords and their values are specified by a keyword and its value in the form `keyword = value`. Comments can also be entered by adding # (Subsequent characters are ignored).

5.3.2 Keywords

- `type`

  Format : str

  Description : The solver type (`OpenMX, QE, VASP, aenet, aenetPyLammps, potts`). When `potts`, the following parameters are not used.

- `path`

  Format : str

  Description : The path to the solver.

- `base_input_dir`

  Format : str or list of str

  Description : The path to the base input file. If multiple calculations are set up in the form of a list, each calculation using each input is performed in turn. For the second and subsequent calculations, the structure from the last step of the previous calculation is used as the initial coordinates, and the energy from the last calculation is used. For example, it is possible to perform a fast structural optimization in the first input file at the expense of accuracy, and then perform the structural optimization in the second and later input files with a higher accuracy setting. Or, in the case of grid vector relaxation, one can run the same input multiple times to reset the computational mesh based on a set plane-wave cutoff.

- `perturb`

  Format : float

  Description : If a structure with good symmetry is input, structure optimization tends to stop at the saddle point. In order to avoid this, an initial structure is formed by randomly displacing each atom in proportion to this parameter. It can also be set to 0.0 or false. Default value = 0.0.

- `ignore_species`

  Format : list

  Description : Specify atomic species to "ignore" in neural network models such as `aenet`. For those that always have an occupancy of 1, it is computationally more efficient to ignore their presence when training and evaluating neural network models.

- `run_scheme` (Only for `sampling.solver`)

  Format : str

  Description : Way to invoke the solver program. For details, please see *Specific notes for first-principles solvers*

- `parallel_level` (Only for `type = "QE"`)

  Format : dict

  Description : How to split parallel cpu resources, i.e., Parallelization levels . Key names are long-form command-line options (without the leading hyphen), that is, `nimage`, `npools`, `nband`, `ntg`, and `ndiag`. Values are the number of parallelization. Only the specified elements will be passed to `pw.x` as command-line options.

- `function` (Only for `type = "user"`)

  Format : str

  Description : Specify a user-defined solver function.

  - The solver function must take a `pymatgen.core.Structure` object as an argument and return a `float` value.
  - When `.` is included, it is assumed that the function is defined in a module, and the module will be automatically imported.
    - For example, if `function = "mypackage.mymodule.myfunction"` is specified, `mypackage.mymodule` is imported and `myfunction` is called.
    - The package is searched from the current directory as well as from the directories specified by the `PYTHONPATH` environment variable.

5.4 [train] section

`abics_train` creates and trains a regression model from configurations to energies. Indeed, `abics_train` uses an external program to train the model. In the current version, only `aenet` is supported as an external program. For software-specific notes (such as input file names), see *Machine learning trainer/calculator-specific notes*.

The input information for `abics_train` is described in the `[trainer]` section. The description of each parameter is as follows. An example is shown as follows:

```
[trainer] # Configure the model trainer.
type = 'aenet'
base_input_dir = '. /aenet_train_input'
```

(continues on next page)

(continued from previous page)

```
exe_command = ['~/git/aenet/bin/generate.x-2.0.4-ifort_serial',
               'srun ~/git/aenet/bin/train.x-2.0.4-ifort_intelmpi']
ignore_species = ["O"]
```

5.4.1 Input Format

Keywords and their values are specified by a keyword and its value in the form `keyword = value`. Comments can also be entered by adding # (Subsequent characters are ignored).

5.4.2 Key words

- `type`

  Format : str

  Description : The trainer to generate the neural network potential (currently only 'aenet').

- `base_input_dir`

  Format : str

  Description : Path of the directory containing the input files that the learner refers to.

- `exe_command`

  Format : list of str

  Description : List of commands to execute; if you use aenet, you need to specify the path to `generate.x` and `train.x`.

- `ignore_species`

  Format : list

  Description : Same as `ignore_species` in [sampling.solver] section. Specify atomic species to "ignore" in neural network models such as `aenet`. For those that always have an occupancy of 1, it is computationally more efficient to ignore their presence when training and evaluating neural network models.

5.5 [observer] section

This section specifies the physical quantity to be calculated. An example is shown as follows:

```
[observer]
[observer.similarity]
reference_structure = './MgAl2O4.vasp'
ignored_species = ["O"]

[[observer.solver]]
name = "magnetization"
type = 'aenet'
path= '~/opt/aenet/bin/predict.x_serial'
base_input_dir = './baseinput_mag'
perturb = 0.0
```

(continues on next page)

(continued from previous page)

```
run_scheme = 'subprocess'
ignore_species = ["O"]
```

5.5.1 Input Format

Keywords and their values are specified by a keyword and its value in the form `keyword = value`. Comments can also be entered by adding # (Subsequent characters are ignored).

5.5.2 Key words

- `[[observer.solver]]`

  This section specifies a physical quantity to be calculated. This section can be specified multiple times. This section is the same as the `sampling.solver` section except for the `name` keyword.

  The quantity with the name `energy` is automatically calculated by using `sampling.solver` .

  - `name`

    Format : str

    Description : The name of the physical quantity to be calculated. After the calculation is completed, the expected value is output as a file named `<name>.dat`.

- `[observer.similarity]`

  "Similarity" is a physical quantity that the ratio of the number of atoms of each element in the same place as the reference state. After the calculation is completed, the expected value is output as a file named `similarity_X.dat` (`X` is the element symbol). When the `reference_structure` keyword is not specified, the similarity is not calculated.

  - `reference_structure`

    Format : str

    Description : Filename of the structure file of the reference state.

  - `ignored_species`

    Format : list

    Description : The atom species to be ignored when calculating the similarity. For example, if you want to ignore the similarity of oxygen atoms, specify `["O"]`.

5.6 [config] section

This section specifies configurations such as alloy coordination. An example is shown as follows:

```
[config]
unitcell = [[8.1135997772, 0.0000000000, 0.0000000000],
            [0.0000000000, 8.1135997772, 0.0000000000],
            [0.0000000000, 0.0000000000, 8.1135997772]]
supercell = [1,1,1]

[[config.base_structure]]
```

(continues on next page)

(continued from previous page)

```
type = "O"
coords = [
    [0.237399980, 0.237399980, 0.237399980],
    [0.762599945, 0.762599945, 0.762599945],
    ...
    [0.262599975, 0.262599975, 0.762599945],
    ]

[[config.defect_structure]]
coords = [
    [0.000000000, 0.000000000, 0.000000000],
    [0.749999940, 0.249999985, 0.499999970],
    ...
    [0.124999993, 0.624999940, 0.124999993],
    ]
[[config.defect_structure.groups]]
name = 'Al'
# species = ['Al']    # default
# coords = [[[0,0,0]]]  # default
num = 16
[[config.defect_structure.groups]]
name = 'Mg'
# species = ['Mg']    # default
# coords = [[[0,0,0]]]  # default
num = 8
```

5.6.1 Input Format

Keywords and their values are specified by a keyword and its value in the form `keyword = value`. Comments can also be entered by adding # (Subsequent characters are ignored).

5.6.2 Key words

In the case of `solver.type != "potts"`

- Specify lattice
  - `unitcell`

    Format : list

    Description : Lattice vector $\mathbf{a}, \mathbf{b}, \mathbf{c}$ by the list format [$\mathbf{a}, \mathbf{b}, \mathbf{c}$] .
  - `supercell`

    Format : list

    Description : The size of super lattice by the list format[$\mathbf{a}, \mathbf{b}, \mathbf{c}$].
  - `init_structure`

    Format : str

Description : The name of structure file (e.g., POSCAR, cif) to be used as the initial structure. If not specified, the initial structure is generated randomly based on `[[config.base_structure]]` and `[[config.defect_structure]]`.

- `constraint_module`

  Format : bool ("true" or "false")

  Description : Whether to apply a constraint condition to configurations. The default value is `false`.

  When it is set to `true`, the constraint condition is given as a user-defined function with the name `constraint_func` in `constraint_module.py`. This function takes structure data of pymatgen.core.Structure type, and returns a boolean value. The module may also contain a function named `constraint_energy` that evaluates energy from the structure data in the `shuffle` operation.

- `constraint`

  Format : str

  Description : The name of the user-defined function used for a constraint condition to configurations. If it is not specified, no constraint is applied. The default is unspecified.

  The constraint condition is given as a function that takes structure data of pymatgen.core.Structure type and returns a boolean value. The parameter `constraint` specifies the name of the function in the form `"module_name.function_name"`. When `constraint_module` is also set to `true`, the value of `constraint` is adopted for the function name.

- `[[config.base_structure]]` section

  `type` and `coords` specify the atomic species that do not move in Monte Carlo calculation and their coordinates. If there are multiple atomic species, specify multiple [[config.base_structure]] sections.

  - `type`

    Format : str

    Description : Atomic specie.

  - `coords`

    Format : list of lists or str

    Description : Coordinates. Specify a list of N elements (number of atoms) arranged in 3 elements representing 3D coordinates, or a string of coordinates arranged in N rows and 3 columns.

- `[[config.defect_structure]]` section

  This sections specifies the lattice coordinates (coords) and atoms (or atom groups) (groups) that can reside on those lattice sites. Monte Carlo sampling is performed on the lattice specified in this section. In Ver. 1.0, conversion tools from POSCAR and cif will be available.

  - `coords`

    Format : list of lists or str

    Description : The coordinates of the lattice sites where atoms reside. A list of N elements (number of atoms) arranged in 3 elements representing 3D coordinates, or a string of coordinates arranged in N rows and 3 columns.

  - [[config.defect_structure.groups]] section

    The atom group information to be updated by Monte Carlo.

* `name`

  Format : str

  Description : The name of atomic group.

* `species`

  Format : list

  Description : The atomic species belonging to the atom group. The default value is a list containing only one specified by `name`. A vacancy can be represented by an empty list `[]`.

* `coords`

  Format : list of lists of lists or str

  Description : The coordinates of each atom in the atom group in each direction of local orientation. *N* (number of atoms) three-element lists, each of which is a list of three elements representing three-dimensional coordinates, is specified as a three-fold list, arranged by orientation. For example, if there are two atoms in the atom group and there are three different orientations, x,y,z, then `coords` can be specified as:

```
coords = [
[ # dir-1
[0.0, 0.0, 0.0], [0.5, 0.0, 0.0]
],
[ # dir-2
[0.0, 0.0, 0.0], [0.0, 0.5, 0.0]
],
[ # dir-3
[0.0, 0.0, 0.0], [0.0, 0.0, 0.5]
],
]
```

  The default value is `[[0.0, 0.0, 0.0]]`, so this keyword can be omitted if there is only one atom in the atom group.

* `relaxation`

  Format : list of lists or str

  Description : Whether to optimize structure (coordinates) or not for each atom and dimension. A list of N elements (number of atoms) with 3 booleans ("true" or "false"), or a string of "true" or "false" arranged in N rows and 3 columns. Default is `["true", "true", "true"]` for all the atoms.

* `magnetization`

  Format : list

  Description : Magnetization (the difference between the number of up and down electrons) for each atom. Default is 0.0 for all the atoms.

* `num`

  Format : int

  Description : The number of atom groups of the type specified in this section.

- `[[config.chemical_potential]]` section

  This section specifies the chemical potentials of the atoms and atom groups for the grand canonical sampling.

- `species`

  Format : str, or list of strs

  Description : Name of atom or atom group, or a list of names of atom groups when a set of atom groups are considered simultaneously.

- `mu`

  Format : float

  Description : The value of chemical potential that corresponds to `species`.

- `[[config.grandcanonical_move]]` section

  This section specifies how the atoms or atom groups are added/removed. It also describes how atoms of one type are replaced by those of another type when such processes are considered.

  - add/remove atoms or atom groups:

    * `species`

      Format : str, or list of strs

      Description : Name of atom or atom group, or a list of names of atom groups when a set of atom groups are considered simultaneously.

  - replace atoms:

    * `from`, `to`

      Format : str, or list of strs

      Description : Names of atoms or atom groups to be replaced are specified in the form of `from A to B`. It also implies the reverse process `from B to A`. The number of atoms of `from` and `to` must be equal, and the atoms are assumed to belong to the same defect sublattice.

  If `grandcanonical_move` is not specified, the addition/removal of `species` of `chemical_potential` are implicitly introduced. Otherwise, only the specified processes may occur.

In the case of `solver.type = "potts"`

- `Q`

  Format : int

  Description : The local degree of freedom of a spin.

- `L`

  Format : List of integers

  Description : Size of a hyper cubic lattice.

5.7 [log] section

This section specifies the log file name and the log level.

5.7.1 Input Format

Keywords and their values are specified by a keyword and its value in the form `keyword = value`. Comments can also be entered by adding # (Subsequent characters are ignored).

5.7.2 Keywords

- `level`

  Format : str

  Description : Logging level. The following levels are available.

  - `debug`
  - `info`
  - `warning`
  - `error`

- `console`

  Format : str

  Description : console output mode.

  - `default` will examine if MPI environment is available or not.
  - `mpi` for parallel environment in which rank numbers are shown in error log.
  - `serial` for serial environment.
  - `none` suppresses console output.

- `console_level`

  Format : str

  Description : Logging level for console output.

- `logfile_path`

  Format : str

  Description : Path to the log file. If not specified, logs will be send only to console. The parent directories will be automatically created if they are not present.

- `logfile_mode`

  Format : str

  Description : MPI log type.

  - `master` will output logs to fiile only from rank=0.
  - `collect` will write messages from all ranks to one file.
  - `workers` will open one log file for each process designated by its rank.
  - `serial` will not consider parallel environment.

- `logfile_level`

  Format : str

  Description : Logging level for log file output.

- `logfile_rank`

  Format : int or list of int

  Description : MPI ranks from which logs are written to file. If not specified, all ranks are taken account of.

CHAPTER

SIX

OUTPUT FILES FORMAT

`RANK` means the rank of process (replica) (`0, 1, ...`).

6.1 RANK/structure.XXX.vasp

The atomic coordinates for each step are saved in the POSCAR file format of VASP. `XXX` in the filename corresponds to the index of the step.

Example:

```
Mg8 Al16 O32
1.0
8.113600 0.000000 0.000000
0.000000 8.113600 0.000000
0.000000 0.000000 8.113600
Al Mg O
16 8 32
direct
0.011208 0.995214 0.998158 Al
0.758187 0.240787 0.499981 Al
... skipped ...
0.746308 0.744706 0.233021 O
0.257199 0.255424 0.771040 O
```

6.2 RANK/minE.vasp

The lowest-energy structure among the samples in this replica.

6.3 RANK/obs.dat

The temperature and the total energy for each step in units of eV.

Example:

```
0      0.1034076       -41690.28269769395
1      0.1034076       -41692.06763035158
2      0.1034076       -41692.06763035158
```

(continues on next page)

(continued from previous page)

```
3       0.1034076       -41691.98205990787
4       0.1034076       -41692.74143710456
```

6.4 RANK/obs_save.npy

The total energy for each step in units of eV in the Numpy binary format. Users can load it as `darray` by using `numpy.load('obs_save.npy')`.

Example:

```
$ python -c "import numpy; print(numpy.load('obs_save.npy'))"
[[-41690.28269769]
 [-41692.06763035]
 [-41692.06763035]
 [-41691.98205991]
 [-41692.7414371 ]]
```

6.5 RANK/kT_hist.npy

The temperature for each step in units of eV in the Numpy binary format. Users can load it as `darray` by using `numpy.load('kT_hist.npy')`.

Example:

```
$ python -c "import numpy; print(numpy.load('kT_hist.npy'))"
[0.1034076 0.1034076 0.1034076 0.1034076 0.1034076]
```

6.6 RANK/Trank_hist.npy

(ONLY for RXMC) The rank (index) of the temperature for each step in the Numpy binary format. Users can load it as `darray` by using `numpy.load('Trank_hist.npy')`.

Example:

```
$ python -c "import numpy; print(numpy.load('Trank_hist.npy'))"
[1 1 1 1 1]
```

6.7 RANK/logweight_hist.npy

(ONLY for PAMC) The logarithm of the Neal-Jarzynski weight for each step in the Numpy binary format.

Example:

```
$ python -c "import numpy; print(numpy.load('logweight_hist.npy'))"
[0 0 0 0 0]
```

6.8 RANK/acceptance_ratio.dat

Acceptance ratio of Monte Carlo steps for each temperature. The first column is temperature and the second column is acceptance ratio (number of accepted / number of trials).

6.9 logZ.dat

The logarithm of the partition function, $\log Z_i/Z_0$ where i is the index of temperature.

- The 1st column is temperature T_i.
- The 2nd and 3rd columns are $\log Z_i/Z_0$ and its error.
- The 4th and 5th columms are $\log Z_i/Z_{i-1}$ and its error.

6.10 <name>.dat

Canonical expectation value $\langle O \rangle$ and statistical error $\sigma[O]$ of an observable O for each temperature. <name> is the name of the observable which is specified by `name` keyword in `[[observer.solver]]` section of the input file.

- The 1st column is temperature T_i.
- The 2nd and 3rd columns are $\langle O \rangle$ and its error.
- The 4th and 5th columns are $\langle O^2 \rangle$ and its error.
- The 6th and 7th columns are fluctuation, $\langle O^2 \rangle - \langle O \rangle^2$ and its error.
  - Note that the heat capacity C is related to the fluctuation of energy as $k_B T^2 C = \left[\langle E^2 \rangle - \langle E \rangle^2\right]$.

CHAPTER

SEVEN

MISCELLANEOUS TOOLS

abICS comes with a few tools for facilitating typical workflows.

- `st2abics` for preparing an abICS input file from a structure file.
- `abicsRXsepT` for postprocessing of replica exchange Monte Carlo run.

The following sections describe how to use each of these tools.

7.1 st2abics

It is sometimes quite tedious to prepare the *[config] section* in abICS *input files*. To facilitate this, we provide the `st2abics` tool, which takes a structure file readable by pymatgen and converts it to an abICS input template with the `[config]` section filled in. An additional control file is required to tell `st2abics` how to break down the original structure file into `config.base_structure` and `config.defect_structure` (see *[config] section* for definitions). The tool is used as follows:

```
$ st2abics -h
usage: st2abics [-h] inputfi structurefi [outfi]

Prepare abICS config from structure file

positional arguments:
  inputfi      toml input file for st2abics
  structurefi  Structure file that can be read by pymatgen Structure.from_file() method
  outfi        Output file to be used as abics input. Defaults to standard output

optional arguments:
  -h, --help   show this help message and exit
```

Examples are provided in `examples/st2abics` and can be run as follows:

```
$ cd examples/st2abics
$ st2abics st2abics_MgAl2O4.toml MgAl2O4.vasp abics_MgAl2O4.toml # spinel
$ st2abics st2abics_CuZn.toml CuZn.vasp abics_CuZn.toml # brass
$ st2abics st2abics_BZY.toml BaZrO3.vasp abics_BZY.toml # Y-doped BaZrO3
```

The resulting files (abics_MgAl2O4.toml, abics_CuZn.toml, and abics_BZY.toml in the above example) can be used as abICS input after filling in the `[mlref]`, `[train]`, `[sampling]` and `[observer]` sections.

7.1.1 Input Format

Examples of st2abics input files can be found in `examples/st2abics` (`st2abics_CuZn.toml`, `st2abics_MgAl2O4.toml`, and `st2abics_BZY.toml` in the above example).

The format is similar to `[config]` section of abICS input file.

7.1.2 Keywords

- `supercell`

  Format : list

  Description : The size of supercell by the list format [$\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$].

- `[[config.base_structure]]` section

  This section specifies how to construct the base_structure that does not exchange atoms between lattice sites during the Monte Carlo calculation.

  - `species`

    Format : list of str

    Description : Atomic species of the base_structure. The corresponding coordinates are extracted automatically from the input structure file.

  - `fix`

    Format : bool

    Description : Whether to disallow local relaxation of the base_structure (true) or not (false).

- `[[config.defect_structure]]` section(s)

  This section specifies the sublattice for configurational sampling. There can be more than one [[config.defect_structure]] section, e.g., one for cations and one for anions.

  - `site_center_species`

    Format : list of str

    Description : The species in the original structure file whose coordinates are used as lattice sites for configurational sampling.

  - `[[config.defect_structure.groups]]` subsection(s) This section specifies the atom groups that reside on the lattice sites for configurational sampling. If not provided, it will be constructed automatically from the original structure file using `site_center_species`.

    * `name`

      Format : str

      Description : The name of atom group.

    * `species`

      Format : list of str

      Description : The atom species belonging to the atom group. The default value is a list containing only one species specified by `name`. Elements that do not appear in the original structure file can also be specified. A vacancy can be represented by an empty list `[]`. As an example, see `st2abics_BZY.toml` in the example directory.

* `coords`

  Format : list of lists of lists or str

  Description : The coordinates of each atom in the atom group for each orientation that the atom group can take (see description for coords *here*). Default value is `[[[0.0, 0.0, 0.0]]]`.

* `num`

  Format : int

  Description : The number of atom groups of the type specified in this section. Make sure to specify the number based on the sites in the supercell, which may be larger than the original structure file read in by st2abics.

7.2 abicsRXsepT

This tool is for reordering the resulting structures and energies at each sampling step of a RXMC run by temperature. It is used after an abICS RXMC run is finished as:

```
$ mpiexec -np NPROCS abicsRXsepT input.toml NSKIP
```

`NPROCS` should be equal to or larger than the number of replicas, and `input.toml` should be replaced by the abICS input file that was used for this run. `NSKIP` is an optional parameter and is used for specifying the number of thermalization steps to skip when calculating the energy averages at each temperature (default value is 0). The results are stored in the `Tseparate` directory, and energy averages vs. temperature are stored in `Tseparate/energies_T.dat`

CHAPTER

EIGHT

ALGORITHM

abICS is designed for combining parallel extended ensemble methods with arbitrary energy calculators. At present, the replica exchange Monte Carlo method and the population annealing Monte Carlo method are implemented.

- Overview of abICS
  - S. Kasamatsu and O. Sugino, J. Phys. Condens. Matter, 31, 085901 (2019).
  - S. Kasamatsu, Y. Motoyama, K. Yoshimi, U. Matsumoto, A. Kuwabara, and T. Ogawa, J. Chem. Phys. 157, 104114 (2022).

8.1 Replica exchange Monte Carlo method

A disadvantage of the widely-used Metropolis Monte Carlo algorithm is that it tends to get stuck in local minima. The replica exchange approach aims to overcome this problem by considering multiple copies, or replicas, of the system under study. The algorithm may be described roughly as follows (see references below for more accurate descriptions). Monte Carlo sampling is performed on each replica independently at varying temperatures. At preset intervals, the temperatures are exchanged according to a Metropolis criterion that essentially assigns lower temperatures to replicas that happen to have lower energies. This allows an efficient sampling of the global configuration space using replicas at higher temperatures and accurate sampling of the local energy landscape at lower temperatures.

In abICS, parameters related to the replica exchange Monte Carlo method are specified in the `[replica]` section of the input file. By setting the lower limit of the replica temperature to $T_s =$ `kTstart` , the upper limit to $T_e =$ `kTend`, and the number of replicas to $N_r =$ `nreplicas`, N_r replica systems with different temperatures ($T_0, T_1, \cdots T_{N_r-1}$) are sampled, where

$$T_i = \frac{T_e - T_s}{N_r - 1} i + T_s.$$

In abICS, using `nprocs_per_replica`, the number of parallel solver processes that performs the calculation on each replica can be specified. The number of Monte Carlo steps is specified by `nsteps`, and the exchange transition probability R for each `RXtrial_frequency` step is defined as

$$R = \exp\left[-\left(\frac{1}{T_i} - \frac{1}{T_k} \right) (E(X_i) - E(X_k)) \right],$$

where X_i is the state for i -th replica system. In abICS, the exchange transition is tried between replicas with adjacent temperatures. The temperature exchange $T_i \leftrightarrow T_k$ is performed with the exchange transition probability R. Physical quantities such as the total energy is measured at each `sample_frequency` step.

- About replica exchange Monte Carlo method
  - K. Hukushima and K. Nemoto, J. Phys. Soc. Japan, 65, 1604 (1996).
  - R. Swendsen and J. Wang, Phys. Rev. Lett. 57, 2607 (1986).

8.2 Population Annealing Monte Carlo

The simulated annealing (SA) method is one of the Monte Carlo method for searching for the ground state. SA performs an ordinally MC sampling under a fixed temperature, and decreases temperature after a given number of MC steps finished. SA first searches a whole space, and then searches narrower regions gradually as temperature decreases. A disadvabtage of SA is that a change of temperature distorts the distribution from equilibrium, and hence thermalization steps are necessary to obtain canonical average of observables for each temperature. The annealed importance sampling (AIS) method overcomes this problem. AIS performs SA on many replicas in parallel, and compensates for deviation of distributions by introducing an extra weight. This weight (Neal-Jarzynski weight) is appropriately defined so that the weighted average of observables over replicas is just the canonical average for each temperature. In AIS, the variance of the NJ weights increases as the simulation progresses, and then only some replicas contribute to the average, in other words, an effective number of replicas decreases. Therefore, the population annealing Monte Carlo (PAMC) method resamples the replicas periodically so that the distribution of replicas is propotional to the NJ weights and resets the weights to 1.

- AIS
  - R. M. Neal, Statistics and Computing 11, 125-139 (2001).
- PAMC
  - K. Hukushima and Y. Iba, AIP Conf. Proc. 690, 200 (2003).

8.3 About configuration and update

Here, the outline of the definition of the configuration in abICS and the update by the Monte Carlo method are explained using Fig. 8.1 as an example.

(a)-(c) are schematic figures of `unitcell`, `base_structure`, and `defect_structure`, where blue, green, and black circles are the atomic types defined by `base_structure`, respectively. The star symbol indicates the location of the defects defined by `defect_structure`. (d) is a schematic figure for specifying the atomic species in `base_structure`. Here, three atomic species of blue, green, and black are defined. How each atom is arranged is defined by `coords` for each atom type. (e) is a schematic figure for specifying the group of atoms to be located at the defect position with `defect_structure`. orange defines a group consisting of four atoms composed of two types of atoms, and purple forms a group of three atoms composed of three types of atoms. These groups are placed at defect points specified by `defect_structure.coords`. The arrangement of atoms in each group can be specified by `coords` in the `defect_structure.groups` section. `defect_structure` can be defined multiple times. Groups of each `defect_structure` will be placed at points of each one. (f) is a schematic figure about the update of the Monte Carlo method. In the update, there are two patterns, one that swaps two atom groups of different type , and the other that changes the orientation within the atom group without changing the arrangement. The type of updates is automatically selected with 1/2 probability. The energy is calculated with the specified solver from the proposed configuration X_{trial} and then the adoption rate $P(X_i \to X_{trial})$ is calculated.

When the grand canonical sampling is turned on by the parameter `sampling.enable_grandcanonical`, the additional update patterns are also introduced in which the atom groups are added from or removed to the reservoir. The numbers of the atom groups are controlled by the chemical potentials `config.chemical_potential` for each atom group or a set of groups simultaneously. The configuration is updated by the Metropolis-Hastings algorithm according to the adoption rate calculated from the differences of the energy and the particle numbers between the original and proposed configurations as

$$P(X_i \to X_{\text{trial}}) = \min\left(1, \Delta W\, e^{-\beta(\Delta E - \mu \Delta N)}\right)$$

where $\Delta W = Q(X_{\text{trial}} \to X_i)/Q(X_i \to X_{\text{trial}})$ denotes the weight factor derived from the proposal distribution $Q(X \to X')$ from a configuration X to X'.

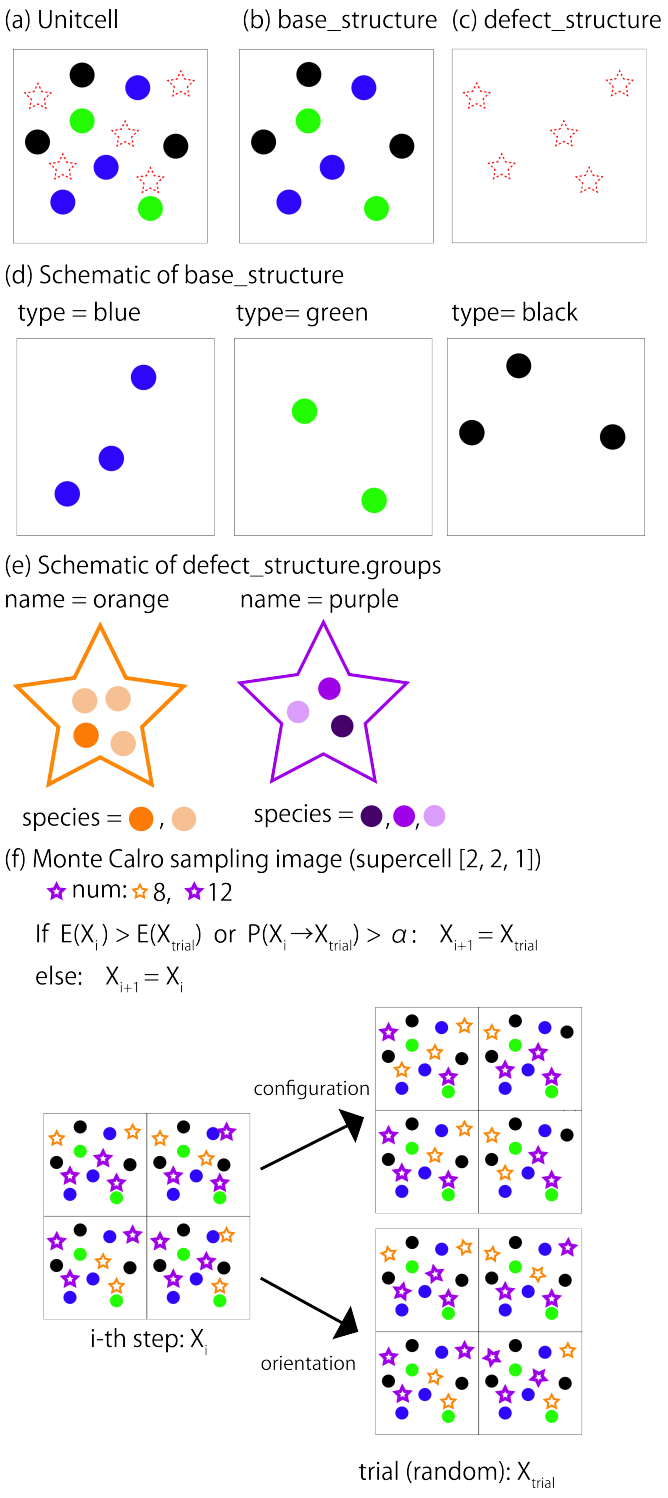


Fig. 8.1: (a)-(e) Definition of lattice in abICS. (f) A schematic of MonteCarlo method. Details are described in the text.

CHAPTER

NINE

ACKNOWLEDGEMENT

The development of this software has been supported by various funding bodies and computer resource providers over the years.

- Priority Issue on Post-K computer (Development of new fundamental technologies for high-efficiency energy creation, conversion/storage and use)
- Joint-use Supercomputer System at Institute for Solid State Physics, the University of Tokyo
- Leading Initiative for Excellent Young Researchers (LEADER) by Ministry of Education, Culture, Science, and Technology (MEXT) of Japan.
- KAKENHI (No. JP18H05519, No. 19K15287) by MEXT
- Core Research for Evolutional Science and Technology (CREST) by Japan Science and Technology Agency (No. JPMJCR15Q3, No. 19K15287)
- New Energy and Industrial Technology Development Organization

We would also like to express our thanks for the support of the "*Project for advancement of software usability in materials science*" of The Institute for Solid State Physics, The University of Tokyo, for the development of abICS.

CHAPTER

TEN

CONTACTS

- About Bugs

  Please report all problems and bugs on the GitHub Issues page

  To resolve bugs early, please follow these guidelines when reporting:

  - Please specify the version of abICS you are using.
  - If there are problems for installation, please inform us about your operating system and the compiler, and include the input / output.
  - If a problem occurs during execution, please provide the input file used for execution and its output.

  Thank you for your cooperation.

- Others

  If you have any questions about topics related to your research that are difficult to consult at Issues on GitHub, please contact the developer team by email.

  E-mail: `abics-dev__at__issp.u-tokyo.ac.jp` (replace _at_ by @)