2DMAT's Documentation

Release 1.0

2DMAT's developer team

Mar 12, 2021

CONTENTS:

CHAPTER

ONE

INTRODUCTION

1.1 What is 2DMAT ?

2DMAT is a framework for applying a search algorithm to a direct problem solver to find the optimal solution. As the standard direct problem solver, the experimental data analysis software for two-dimensional material structure analysis is prepared. The direct problem solver gives the deviation between the experimental data and the calculated data obtained under the given parameters such as atomic positions as a loss function used in the inverse problem. The optimal parameters are estimated by minimizing the loss function using a search algorithm. For further use, the original direct problem solver or the search algorithm can be defined by users. In the current version, for solving a direct problem, 2DMAT offers the wrapper of the solver for the total-reflection high-energy positron diffraction (TRHEPD) experiment[1, 2]. As algorithms, it offers the Nelder-Mead method[3], the grid search method[4], the Bayesian optimization method[5], and the replica exchange Monte Carlo method[6]. In the future, we plan to add other direct problem solvers and search algorithms in 2DMAT.

[1] As a review, see Y. Fukaya, et al., J. Phys. D: Appl. Phys. 52, 013002 (2019).

[2] This software has been developed by T. Hanada in Tohoku University. T. Hanada, H. Daimon, and S. Ino, Phys. Rev. B 51, 13320 (1995).

[3] K. Tanaka, T. Hoshi, I. Mochizuki, T. Hanada, A. Ichimiya, and T. Hyodo, Acta. Phys. Pol. A 137, 188 (2020).

[4] K. Tanaka, I. Mochizuki, T. Hanada, A. Ichimiya, T. Hyodo, and T. Hoshi, JJAP Conf. Series, in press, arXiv:2002.12165.

[5] The python package PHYSBO is used for Bayesian optimization.

[6] K. Hukushima and K. Nemoto, J. Phys. Soc. Japan, 65, 1604 (1996), R. Swendsen and J. Wang, Phys. Rev. Lett. 57, 2607 (1986).

1.2 License

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. We hope that you cite the following reference when you publish the results using 2DMAT:

Kazuyuki Tanaka, Takeo Hoshi, Izumi Mochizuki, Takashi Hanada, Ayahiko Ichimiya, Toshio Hyodo, Acta. Phys. Pol. A 137(3) 188 - 192 2020

1.3 Version Information

- v1.0.0: 2021-03-12
- v0.1.0: 2021-02-08

1.4 Main developers

2DMAT has been developed by following members.

- v0.1.0 -
  - Y. Motoyama (The Institute for Solid State Physics, The University of Tokyo)
  - K. Yoshimi (The Institute for Solid State Physics, The University of Tokyo)
  - T. Hoshi (Department of Applied Mathematics and Physics, Tottori University)

CHAPTER

TWO

INSTALL OF PY2DMAT

2.1 Prerequisites

- Python3 (>=3.6)
  - **The following Python packages are required.**
    - toml
    - numpy
  - Optional packages
    - mpi4py (required for grid search)
    - scipy (required for Nelder-Mead method)
    - physbo (>=0.3, required for Baysian optimization)

2.2 How to download and install

You can install the `py2dmat` python package and the `py2dmat` command using the method shown below.

- Installation using PyPI (recommended)
  - `python3 -m pip install py2dmat`
    - `--user` option to install locally (`$HOME/.local`)
    - If you use `py2dmat[all]`, optional packages will be installed at the same time.
- Installation from source code
  1. `git clone https://github.com/issp-center-dev/2DMAT`
  2. `python3 -m pip install ./2DMAT`
     - The `pip` version must be 19 or higher (can be updated with `python3 -m pip install -U pip`).
- Download the sample files
  - Sample files are included in the source code.
  - `git clone https://github.com/issp-center-dev/2DMAT`

Note that among the direct problem solvers used in `py2dmat`, the following solver must be installed separately:

- TRHEPD forward problem solver (`sim-trhepd-rheed`)

Please refer to the tutorials of each solver for installation details.

2.3 How to run

In `py2dmat` , the analysis is done by using a predefined optimization algorithm `Algorithm` and a direct problem solver `Solver`

```
$ py2dmat input.toml
```

See *Search algorithms* for the predefined `Algorithm` and solver/input for the `Solver`.

If you want to prepare the `Algorithm` or `Solver` by yourself, use the `py2dmat` package. See *(For developers) User-defined algorithm and solver* for details.

2.4 How to uninstall

Please type the following command:

```
$ python3 -m pip uninstall py2dmat
```

CHAPTER

THREE

INPUT FILE

As the input file format, TOML format is used. The input file consists of the following four sections.

- `base`
  - Specify the basic parameters about `py2dmat` .
- `solver`
  - Specify the prarameters about `Solver` .
- `algorithm`
  - Specify the prarameters about `Algorithm` .
- `runner`
  - Specify the parameters about `Runner` .

3.1 [base] section

- `dimension`

  Format: Integer

  Description: Dimension of the search space (number of parameters to search)

- `output_dir`

  Format: string (default: The directory where the program was executed)

  Description: Name of the directory to output the results.

3.2 [solver] section

The `name` determines the type of solver. Each parameter is defined for each solver.

- `name`

  Format: String

  Description: Name of the solver. The following solvers are available.

  - `sim-trhepd-rheed` : Solver to calculate Total-reflection high energy positron diffraction (TRHEPD) or Reflection High Energy Electron Diffraction (RHEED) intensities.
  - `analytical` : Solver to provide analytical solutions (mainly used for testing).

See *Direct Problem Solver* for details of the various solvers and their input/output files.

3.3 [algorithm] section

The `name` determines the type of algorithm. Each parameter is defined for each algorithm.

- `name`

  Format: String

  Description: Algorithm name. The following algorithms are available.

  - `minsearch` : Minimum value search using Nelder-Mead method
  - `mapper` : Grid search
  - `exchange` : Replica Exchange Monte Carlo
  - `bayes` : Bayesian optimization

- `seed`

  Format: Integer

  Description: A parameter to specify seeds of the pseudo-random number generator used for random generation of initial For each MPI process, the value of `seed + mpi_rank * seed_delta` is given as seeds. If omitted, the initialization is done by the Numpy's prescribed method.

- `seed_delta`

  Format: Integer (default: 314159)

  Description: A parameter to calculate the seed of the pseudo-random number generator for each MPI process. For details, see the description of `seed`.

See *Search algorithms* for details of the various algorithms and their input/output files.

3.4 [runner] section

This section sets the configuration of `Runner`, which bridges `Algorithm` and `Solver`. It has a subsection `log`

3.4.1 [log] section

Settings related to logging of solver calls.

- `filename`

  Format: String (default: "runner.log")

  Description: Name of log file.

- `interval`

  Format: Integer (default: 0)

  Description: The log will be written out every time solver is called `interval` times. If the value is less than or equal to 0, no log will be written.

- `write_result`

  Format: Boolean (default: false)

  Description: Whether to record the output from solver.

- `write_input`

  Format: Boolean (default: false)

  Description: Whether to record the input to solver.

CHAPTER

FOUR

OUTPUT FILES

See *Direct Problem Solver* and *Search algorithms* for the output files of each `Solver` and `Algorithm`.

4.1 Common file

4.1.1 `time.log`

The total time taken for the calculation for each MPI rank is outputted. These files will be output under the subfolders of each rank respectively. The time taken to pre-process the calculation, the time taken to compute, and the time taken to post-process the calculation are listed in the `prepare` , `run` , and `post` sections.

The following is an example of the output.

```
#prepare
 total = 0.007259890999989693
#run
 total = 1.3493346729999303
 - file_CM = 0.0009563499997966574
 - submit = 1.3224223930001244
#post
 total = 0.000595873999941432
```

4.1.2 `runner.log`

The log information about solver calls for each MPI rank is outputted. These files will be output under the subfolder of each rank. The output is only available when the `runner.log.interval` parameter is a positive integer in the input.

- The first column is the serial number of the solver call.
- The second column is the time elapsed since the last solver call.
- The third column is the time elapsed since the start of the calculation.

The following is an example of the output.

```
# $1: num_calls
# $2: elapsed_time_from_last_call
# $3: elapsed_time_from_start

1 0.0010826379999999691 0.0010826379999999691
2 6.96760000000185e-05 0.0011523139999999876
```

(continues on next page)

(continued from previous page)

```
3 9.67080000000009e-05 0.0012490219999999885
4 0.00011765699999999324 0.0013666789999999818
5 4.965899999997969e-05 0.0014163379999999615
6 8.666900000003919e-05 0.0015030070000000006
  ...
```

CHAPTER

FIVE

DIRECT PROBLEM SOLVER

Direct problem solver `Solver` calculates the function to be optimized $f(x)$ at the search parameter x.

5.1 `analytical` solver

`analytical` is a `Solver` that computes a predefined benchmark function $f(x)$ for evaluating the performance of search algorithms.

5.1.1 Input parameters

The `funtion_name` parameter in the `solver` section specifies the function to use.

- `function_name`

  Format: string

  Description: Function name. The following functions are available.

  - `quadratics`
    - Quadratic function

$$f(\vec{x}) = \sum_{i=1}^{N} x_i^2$$

    - The optimized value $f(\vec{x}^*) = 0 \quad (\forall_i x_i^* = 0)$
  - `rosenbrock`
    - Rosenbrock function

$$f(\vec{x}) = \sum_{i=1}^{N-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

    - The optimized value $f(\vec{x}^*) = 0 \quad (\forall_i x_i^* = 1)$
  - `ackley`
    - Ackley function

$$f(\vec{x}) = 20 + e - 20\exp\left[-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right] - \exp\left[\frac{1}{N}\cos\left(2\pi x_i\right)\right]$$

    - The optimized value $f(\vec{x}^*) = 0 \quad (\forall_i x_i^* = 0)$

- himmerblau
  - Himmerblau function

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

  - The optimized value $f(3, 2) = f(-2.805118, 3.131312) = f(-3.779310, -3.283186) = f(3.584428, -1.848126) = 0$

5.2 sim-trhepd-rheed solver

sim-trhepd-rheed is a Solver that uses `sim-trhepd-rheed`_ to calculate the diffraction rocking curve from the atomic position x and returns the deviation from the experimental rocking curve as $f(x)$.

5.2.1 Preparation

You will need to install `sim-trhepd-rheed`_ beforehand.

1. Download the source code from the official sim-trhepd-rheed website.
2. Move to sim-trhepd-rheed/src folder and make bulk.exe and surf.exe by using make.

Before running py2dmat, run bulk.exe to create the bulk data. The surf.exe is called from py2dmat.

5.2.2 Input parameters

Input parameters can be specified in subcsections config, post, param, reference in solver section.

[config] section

- surface_exec_file

  Format: string (default: "surf.exe")

  Description: Path to sim-trhepd-rheed surface reflection solver surf.exe.

- surface_input_file

  Format: string (default: "surf.txt")

  Description: Input file for surface structure.

- bulk_output_file

  Format: string (default: "bulkP.b")

  Description: Output file for bulk structure.

- surface_output_file

  Format: string (default: "surf-bulkP.s")

  Description: Output file for surface structure.

- calculated_first_line

  Format: integer (default: 5)

  Description: One of the parameters that specifies the range of output files to be read, calculated by the solver. This parameter specifies the first line to be read.

- `calculated_last_line`

  Format: integer (default: 60)

  Description: One of the parameters that specifies the range of output files to be read, calculated by the solver. This parameter specifies the last line to be read.

- `row_number`

  Format: integer (default: 8)

  Description: One of the parameters that specifies the range of output files to be read, calculated by the solver. This parameter specifies the column to be read.

[post] section

- `normalization`

  Format: string ("TOTAL" or "MAX", default: "TOTAL")

  Description: This parameter specifies whether the R-value is normalized by the sum of the whole values or by the maximum value.

- `Rfactor_type`

  Format: string ("A" or "B", default: "A")

  Description: This parameter specifies how to calculate the R-factor. "A" means the normal method, "B" means Pendry's R-factor is used.

- `omega`

  Format: float (default: 0.5)

  Description: This parameter specifies the half-width of convolution.

[param] section

- `string_list`

  Format: list of string. The length should match the value of dimension (default: ["value_01", "value_02"]).

  Description: List of placeholders to be used in the reference template file to create the input file for the solver. These strings will be replaced with the values of the parameters being searched for.

- `degree_max`

  Format: float (default: 6.0)

  Description: Maximum angle (in degrees)

[reference] section

- path

  Format: string (default: experiment.txt)

  Description: Path to the experimental data file.

- first

  Format: integer (default: 1)

  Description: One of the parameters that specify the range of experimental data files to be read. This parameter specifies the first line of the experimental file to be read.

- last

  Format: integer (default: 56)

  Description: One of the parameters that specify the range of experimental data files to be read. This parameter specifies the last line of the experimental file to be read.

5.2.3 Reference file

Input template file

The input template file template.txt is a template for creating an input file for surf.exe. The parameters to be moved in py2dmat (such as the atomic coordinates you want to find) should be replaced with the appropriate string, such as value_*. The strings to be used are specified by string_list in the [solver] - [param] section of the input file for the solver. An example template is shown below.

```
2                                       ,NELMS,  -------- Ge(001)-c4x2
32,1.0,0.1                              ,Ge Z,da1,sap
0.6,0.6,0.6                             ,BH(I),BK(I),BZ(I)
32,1.0,0.1                              ,Ge Z,da1,sap
0.4,0.4,0.4                             ,BH(I),BK(I),BZ(I)
9,4,0,0,2, 2.0,-0.5,0.5                  ,NSGS,msa,msb,nsa,nsb,dthick,DXS,DYS
8                                       ,NATM
1, 1.0, 1.34502591  1       value_01    ,IELM(I),ocr(I),X(I),Y(I),Z(I)
1, 1.0, 0.752457792 1       value_02
2, 1.0, 1.480003343 1.465005851     value_03
2, 1.0, 2   1.497500418     2.281675
2, 1.0, 1   1.5     1.991675
2, 1.0, 0   1       0.847225
2, 1.0, 2   1       0.807225
2, 1.0, 1.009998328 1       0.597225
1,1                                     ,(WDOM,I=1,NDOM)
```

In this case, value_01, value_02, and value_03 are the parameters to be moved in py2dmat.

Target file

This file (`experiment.txt`) contains the data to be targeted. The first column contains the angle, and the second column contains the calculated value of the reflection intensity multiplied by the weight. An example of the file is shown below.

```
0.100000 0.002374995
0.200000 0.003614789
0.300000 0.005023215
0.400000 0.006504978
0.500000 0.007990674
0.600000 0.009441623
0.700000 0.010839445
0.800000 0.012174578
0.900000 0.013439485
1.000000 0.014625579
...
```

5.2.4 Output file

For `sim-trhepd-rheed`, the files output by `surf.exe` will be output in the `Log%%%%%` folder under the folder with the rank number. This section describes the own files that are output by this solver.

stdout

It contains the standard output of `surf.exe`. An example is shown below.

```
bulk-filename (end=e) ? :
bulkP.b
structure-filename (end=e) ? :
surf.txt
output-filename :
surf-bulkP.s
```

RockingCurve.txt

This file is located in the `Log%%%%%` folder. The first line is the header, and the second and subsequent lines are the angle, convoluted calculated/experimental values, normalized calculated/experimental values, and raw calculated values in that order. An example is shown below.

```
#degree convolution_I_calculated I_experiment convolution_I_calculated(normalized) I_
↪experiment(normalized) I_calculated
0.1 0.0023816127859192407 0.002374995 0.004354402952499057 0.005364578226620574 0.
↪001722
0.2 0.003626530149456865 0.003614789 0.006630537795012198 0.008164993342397588 0.
↪003397
0.3 0.00504226607469267 0.005023215 0.009218987407498791 0.011346310125551366 0.005026
0.4 0.006533558304296079 0.006504978 0.011945579793136154 0.01469327865677437 0.006607
0.5 0.00803056955158873 0.007990674 0.014682628499657693 0.018049130948243314 0.008139
0.6 0.009493271317558538 0.009441623 0.017356947736613827 0.021326497600946535 0.00962
0.7 0.010899633015118851 0.010839445 0.019928258053867838 0.024483862338931763 0.01105
...
```

CHAPTER

SIX

SEARCH ALGORITHMS

`py2dmat` searches the parameter space $\mathbf{X} \ni x$ by using the search algorithm `Algorithm` and the result of `Solver` $f(x)$. In this section, the search algorithms implemented in `py2dmat` are described.

6.1 Nelder-Mead method `minsearch`

When `minsearch` is selcted, the optimization by the Nelder-Mead method (a.k.a. downhill simplex method) will be done. In the Nelder-Mead method, the dimension of the parameter space is D, and the optimal solution is searched by systematically moving pairs of $D + 1$ coordinate points according to the value of the objective function at each point.

An important hyperparameter is the initial value of the coordinates. Although it is more stable than the simple steepest descent method, it still has the problem of being trapped in the local optimum solution, so it is recommended to repeat the calculation with different initial values several times to check the results.

In 2DMAT, the Scipy's function `scipy.optimize.minimize(method="Nelder-Mead")` is used. For details, see the official document .

6.1.1 Preparation

You will need to install scipy .:

```
python3 -m pip install scipy
```

6.1.2 Input parameters

It has subsections `param` and `minimize`.

[`param`] section

- `initial_list`

  Format: List of float. The length should match the value of dimension.

  Description: Initial value of the parameter. If not defined, it will be initialized uniformly and randomly.

- `unit_list`

  Format: List of float. The length should match the value of dimension.

Description: Units for each parameter. In the search algorithm, each parameter is divided by each of these values to perform a simple dimensionless and normalization. If not defined, the value is 1.0 for all dimensions.

- `min_list`

  Format: List of float. The length should match the value of dimension.

  Description: The minimum value the parameter can take.

- `max_list`

  Format: List of float. The length should match the value of dimension.

  Description: The maximum value the parameter can take.

`[minimize]` section

Set the hyperparameters for the Nelder-Mead method. See the documentation of scipy.optimize.minimize for details.

- `initial_scale_list`

  Format: List of float. The length should match the value of dimension.

  Description: The difference value that is shifted from the initial value in order to create the initial simplex for the Nelder-Mead method. The `initial_simplex` is given by the sum of `initial_list` and the dimension of the `initial_list` plus one component of the `initial_scale_list`. If not defined, scales at each dimension are set to 0.25.

- `xatol`

  Format: Float (default: 1e-4)

  Description: Parameters used to determine convergence of the Nelder-Mead method.

- `fatol`

  Format: Float (default: 1e-4)

  Description: Parameters used to determine convergence of the Nelder-Mead method.

- `maxiter`

  Format: Integer (default: 10000)

  Description: Maximum number of iterations for the Nelder-Mead method.

- `maxfev`

  Format: Integer (default: 100000)

  Description: Maximum number of times to evaluate the objective function.

6.1.3 Output files

`SimplexData.txt`

Outputs information about the process of finding the minimum value. The first line is a header, the second and subsequent lines are step, the values of variables defined in `string_list` in the `[solver]` - `[param]` sections of the input file, and finally the value of the function.

The following is an example of the output.

```
#step z1 z2 z3 R-factor
0 5.25 4.25 3.5 0.015199251773721183
1 5.25 4.25 3.5 0.015199251773721183
2 5.229166666666666 4.3125 3.645833333333333 0.013702918021532375
3 5.225694444444445 4.40625 3.5451388888888884 0.012635279378225261
4 5.179976851851851 4.348958333333334 3.5943287037037033 0.006001660077530159
5 5.179976851851851 4.348958333333334 3.5943287037037033 0.006001660077530159
```

res.txt

The value of the final objective function and the value of the parameters at that time are described. The objective function is listed first, followed by the values of the variables defined in `string_list` in the `[solver]` - `[param]` sections of the input file, in that order.

The following is an example of the output.

```
fx = 7.382680568652868e-06
z1 = 5.230524973874179
z2 = 4.370622919269477
z3 = 3.5961444501081647
```

6.2 Direct parallel search `mapper`

`mapper_mpi` is an algorithm to search for the minimum value by computing $f(x)$ on all the candidate points in the parameter space prepared in advance. In the case of MPI execution, the set of candidate points is divided into equal parts and automatically assigned to each process to perform trivial parallel computation.

6.2.1 Preparation

For MPI parallelism, you need to install mpi4py.:

```
python3 -m pip install mpi4py
```

6.2.2 Input parameters

[param] section

In this section, the search parameter space is defined.

If `mesh_path` is defined, it is read from a mesh file. In the mesh file, one line defines one point in the parameter space, the first column is the data number, and the second and subsequent columns are the coordinates of each dimension.

If `mesh_path` is not defined, `min_list`, `max_list`, and `num_list` are used to create an evenly spaced grid for each parameter.

- `mesh_path`

  Format: String

  Description: Path to the mesh definition file.

- `min_list`

  Format: List of float. The length should match the value of dimension.

  Description: The minimum value the parameter can take.

- `max_list`

  Format: List of float.The length should match the value of dimension.

  Description: The maximum value the parameter can take.

- `num_list`

  Format: List of integer. The length should match the value of dimension.

  Description: The number of grids the parametar can take at each dimension.

6.2.3 Refernce file

Mesh definition file

Define the grid space to be explored in this file. The first column is the index of the mesh, and the second and subsequent columns are the values of variables defined in `string_list` in the `[solver.param]` section.

Below, a sample file is shown.

```
1 6.000000 6.000000
2 6.000000 5.750000
3 6.000000 5.500000
4 6.000000 5.250000
5 6.000000 5.000000
6 6.000000 4.750000
7 6.000000 4.500000
8 6.000000 4.250000
9 6.000000 4.000000
...
```

6.2.4 Output file

`ColorMap.txt`

This file contains the candidate parameters for each mesh and the `R-factor` at that time. The mesh data is listed in the order of the variables defined in `string_list` in the `[solver]` - `[param]` sections of the input file, and the value of the `R-factor` is listed last.

Below, output example is shown.

```
6.000000 6.000000 0.047852
6.000000 5.750000 0.055011
6.000000 5.500000 0.053190
6.000000 5.250000 0.038905
6.000000 5.000000 0.047674
6.000000 4.750000 0.065919
6.000000 4.500000 0.053675
6.000000 4.250000 0.061261
6.000000 4.000000 0.069351
```

(continues on next page)

(continued from previous page)

```
6.000000 3.750000 0.071868
...
```

6.3 Bayse optimization `bayes`

`bayes` is an `Algorithm` that uses Bayesian optimization to perform parameter search. The implementation is based on PHYSBO.

6.3.1 Preparation

You will need to install PHYSBO beforehand.:

```
python3 -m pip install physbo
```

If mpi4py is installed, MPI parallel computing is possible.

6.3.2 Input parameters

`[algorithm.param]` section

In this section, the search parameter space is defined.

If `mesh_path` is defined, it will be read from a mesh file. In a mesh file, one line gives one point in the parameter space, the first column is the data number, and the second and subsequent columns are the coordinates of each dimension.

If `mesh_path` is not defined, `min_list`, `max_list`, and `num_list` are used to create an evenly spaced grid for each parameter.

- `mesh_path`

  Format: String

  Description: The path to a reference file that contains information about the mesh data.

- `min_list`

  Format: List of float. The length should match the value of dimension.

  Description: The minimum value the parameter can take.

- `max_list`

  Format: List of float.The length should match the value of dimension.

  Description: The maximum value the parameter can take.

- `num_list`

  Format: List of integer. The length should match the value of dimension.

  Description: The number of grids the parametar can take at each dimension.

[algorithm.bayes] section

The hyper parameters are defined.

- random_max_num_probes

  Format: Integer (default: 20)

  Description: Number of random samples to be taken before Bayesian optimization (random sampling is needed if parameters and scores are not available at the beginning).

- bayes_max_num_probes

  Format: Integer (default: 40)

  Description: Number of times to perform Bayesian optimization.

- score

  Format: String (default: TS)

  Description: Parameter to specify the score function. EI (expected improvement), PI (probability of improvement), and TS (Thompson sampling) can be chosen.

- interval

  Format: Integer (default: 5)

  Description: The hyperparameters are learned at each specified interval. If a negative value is specified, no hyperparameter learning will be performed. If a value of 0 is specified, hyperparameter learning will be performed only in the first step.

- num_rand_basis

  Format: Integer (default: 5000)

  Description: Number of basis functions; if 0 is specified, the normal Gaussian process is performed without using the Bayesian linear model.

6.3.3 Reference file

Mesh definition file

Define the grid space to be explored in this file. The first column is the index of the mesh, and the second and subsequent columns are the values of variables defined in string_list in the [solver.param] section.

Below, a sample file is shown.

```
1 6.000000 6.000000
2 6.000000 5.750000
3 6.000000 5.500000
4 6.000000 5.250000
5 6.000000 5.000000
6 6.000000 4.750000
7 6.000000 4.500000
8 6.000000 4.250000
9 6.000000 4.000000
...
```

6.3.4 Output files

BayesData.txt

At each step of the optimization process, the values of the parameters and the corresponding objective functions are listed in the order of the optimal parameters so far and the searched parameters at that step.

```
#step z1 z2 R-factor z1_action z2_action R-factor_action
0 4.75 4.5 0.05141906746102885 4.75 4.5 0.05141906746102885
1 4.75 4.5 0.05141906746102885 6.0 4.75 0.06591878368102033
2 5.5 4.25 0.04380131351780189 5.5 4.25 0.04380131351780189
3 5.0 4.25 0.02312528177606794 5.0 4.25 0.02312528177606794
...
```

6.3.5 Algorithm Description

Bayesian optimization (BO) is an optimization algorithm that uses machine learning as an aid, and is particularly powerful when it takes a long time to evaluate the objective function.

In BO, the objective function $f(\vec{x})$ is approximated by a model function (often a Gaussian process) $g(\vec{x})$ that is quick to evaluate and easy to optimize. The g is trained to reproduce well the value of the objective function $\{\vec{x}_i\}_{i=1}^N$ at some suitably predetermined points (training data set) $\{f(\vec{x}_i)\}_{i=1}^N$.

At each point in the parameter space, we propose the following candidate points for computation $\vec{x}_{N+1}$, where the expected value of the trained $g(\vec{x})$ value and the "score" (acquition function) obtained from the error are optimal. The training is done by evaluating $f(\vec{x}_{N+1})$, adding it to the training dataset, and retraining g. After repeating these searches, the best value of the objective function as the optimal solution will be returned.

A point that gives a better expected value with a smaller error is likely to be the correct answer, but it does not contribute much to improving the accuracy of the model function because it is considered to already have enough information. On the other hand, a point with a large error may not be the correct answer, but it is a place with little information and is considered to be beneficial for updating the model function. Selecting the former is called "utilization," while selecting the latter is called "exploration," and it is important to balance both. The definition of "score" defines how to choose between them.

In 2DMAT, we use PHYSBO as a library for Bayesian optimization. PHYSBO, like `mapper_mpi`, computes a "score" for a predetermined set of candidate points, and proposes an optimal solution. MPI parallel execution is possible by dividing the set of candidate points. In addition, we use a kernel that allows us to evaluate the model function and thus calculate the "score" with a linear amount of computation with respect to the number of training data points N. In PHYSBO, "expected improvement (EI)", "probability of improvement (PI)", and "Thompson sampling (TS)" are available as "score" functions.

6.4 Replica exchange Monte Carlo `exchange`

`exchange` explores the parameter space by using the replica exchange Monte Carlo (RXMC) method.

6.4.1 Preparation

mpi4py should be installed.

```
python3 -m pip install mpi4py
```

6.4.2 Input parameters

This has two subsections `algorithm.param` and `algorithm.exchange` .

[algorithm.param]

- `initial_list`

  Format: List of float. Length should be equal to `dimension`.

  Description: Initial value of parameters. If not defined, these will be initialize randomly.

- `unit_list`

  Format: List of float. Length should be equal to `dimension`.

  Description: Unit length of each parameter. `Algorithm` makes parameters dimensionless and normalized by dividing these by `unit_list`. If not defined, each component will be 1.0.

- `min_list`

  Format: List of float. Length should be equal to `dimension`.

  Description: Minimum value of each parameter.

- `max_list`

  Format: List of float. Length should be equal to `dimension`.

  Description: Maximum value of each parameter.

[algorithm.exchange]

- `numsteps`

  Format: Integer

  Description: The number of Monte Carlo steps.

- `numsteps_exchange`

  Format: Integer

  Description: The number of interval Monte Carlo steps between replica exchange.

- `Tmin`

  Format: Float (default: 0.1)

  Description: The minimum value of the "temperature".

- `Tmax`

  Format: Float (default: 10.0)

  Description: The maximum value of the "temperature".

- `Tlogspace`

  Format: Boolean (default: true)

  Description: Whether to assign "temperature" to replicas equally spaced in the logarithmic space or not.

6.4.3 Output files

`RANK/trial.txt`

This file stores the suggested parameters and the corresponding value returned from the solver for each replica. The first column is the index of the MC step. The second column is the temperature of the replica. The third column is the value of the solver. The remaining columns are the coordinates.

Exapmle:

```
# step T fx z1 z2
0 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
1 0.004999999999999999 0.0758494287185766 2.811346329442423 3.691101784194861
2 0.004999999999999999 0.08566823949124412 3.606664760390988 3.2093903670436497
3 0.004999999999999999 0.06273922648753057 4.330900869594549 4.311333132184154
```

`RANK/result.txt`

This file stores the sampled parameters and the corresponding value returned from the solver for each replica. This has the same format as `trial.txt`.

```
# step T fx z1 z2
0 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
1 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
2 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
3 0.004999999999999999 0.06273922648753057 4.330900869594549 4.311333132184154
```

`best_result.txt`

The optimal value of the solver and the corresponding parameter among the all samples.

```
nprocs = 4
rank = 2
step = 65
fx = 0.008233957976993406
z1 = 4.221129370933539
z2 = 5.139591716517661
```

Algorithm

6.4.4 Markov chain Monte Carlo

The Markov chain Monte Carlo (MCMC) sampling explores the parameter space by moving walkers $\vec{x}$ stochastically according to the weight function $W(\vec{x})$. For the weight function, the Boltzmann factor $W(\vec{x}) = e^{-f(\vec{x})/T}$ is generally adopted, where $T > 0$ is the "temperature." It is impossible in the many cases, unfortunately, to sample walkers according to W directly. Insteadly, the MCMC method moves walkers slightly and generates a time series $\{\vec{x}_t\}$ such that the distribution of the walkers obeys W . Let us call the transision probability from $\vec{x}$ to $\vec{x}'$ as $p(\vec{x}'|\vec{x})$. When p is determined by the following condition ("the balance condition")

$$W(\vec{x}') = \sum_{\vec{x}} p(\vec{x}'|\vec{x})W(\vec{x}),$$

the distribution of the generated time series $\{\vec{x}_t\}$ will converges to $W(\vec{x})$[1]. Practically, the stronger condition ("the detailed balance condition")

$$p(\vec{x}|\vec{x}')W(\vec{x}') = W(\vec{x})p(\vec{x}'|\vec{x})$$

is usually imposed. The detailed balance condition returns to the balance condition by taking the summation of $\vec{x}$.

2DMAT adopts the Metropolis-Hasting (MH) method for solving the detailed balance condition. The MH method splits the transition process into the suggestion process and the acceptance process.

1. Generate a candidate $\vec{x}$ with the suggestion probability $P(\vec{x}|\vec{x}_t)$.
    - As P, use a simple distribution such as the normal distribution with centered at x.
2. Accept the candidate $\vec{x}$ with the acceptance probability $Q(\vec{x}|\vec{x}_t)$.
    - If accepted, let $\vec{x}_{t+1}$ be *vec{x}*.
    - Otherwise, let $\vec{x}_{t+1}$ be *vec{x}_t*.

The whole transision probability is the product of these two ones, $p(\vec{x}|\vec{x}_t) = P(\vec{x}|\vec{x}_t)Q(\vec{x}|\vec{x}_t)$. The acceptance probability $Q(\vec{x}|\vec{x}_t)$ is defined as

$$Q(\vec{x}|\vec{x}_t) = \min\left[1, \frac{W(\vec{x})P(\vec{x}_t|\vec{x})}{W(\vec{x}_t)P(\vec{x}_t|\vec{x})}\right].$$

It is easy to verify that the detailed balance condition is satisfied by substituting it into the detailed balance condition equation.

When adopting the Boltzmann factor for the weight and a symmetry distribution $P(\vec{x}|\vec{x}_t) = P(\vec{x}_t|\vec{x})$ for the suggestion probability, the acceptance probability Q will be the following simple form:

$$Q(\vec{x}|\vec{x}_t) = \min\left[1, \frac{W(\vec{x})}{W(\vec{x}_t)}\right] = \min\left[1, \exp\left(-\frac{f(\vec{x}) - f(\vec{x}_t)}{T}\right)\right].$$

By saying $\Delta f = f(\vec{x}) - f(\vec{x}_t)$ and using the fact $Q = 1$ for $\Delta f \leq 0$, the procedure of MCMC with the MH algorithm is the following:

1. Choose a candidate from near the current position and calculate f and Δf.
2. If $\Delta f \leq 0$, that is, the walker is descending, accept it.
3. Otherwise, accept it with the probability $Q = e^{-\Delta f/T}$.
4. Repeat 1-3.

The solution is given as the point giving the minimum value of $f(\vec{x})$. The third process of the above procedure endures that walkers can climb over the hill with a height of $\Delta f \sim T$, the MCMC sampling can escape from local minima.

[1] To be precisely, the non-periodicality and the ergodicity are necessary for convergence.

6.4.5 Replica exchange Monte Carlo

The "temperature" T is one of the most important hyper parameters in the MCMC sampling. The MCMC sampling can climb over the hill with a height of T but cannot easily escape from the deeper valley than T. It is why we should increase the temperature in order to avoid stuck to local minima. On the other hand, since walkers cannot see the smaller valleys than T, the precision of the obtained result $\min f(\vec{x})$ becomes about T, and it is necessary to decrease the temperature in order to achieve more precise result. This dilemma leads us that we should tune the temperature carefully.

One of the ways to overcome this problem is to update temperature too. For example, simulated annealing decreases temperature as the iteration goes. Another algorithm, simulated tempering, treats temperature as another parameter to be sampled, not a fixed hyper parameter, and update temperature after some iterations according to the (detailed) balance condition. Simulated tempering studies the details of a valley by cooling and escapes from a valley by heating. Replica exchange Monte Carlo (RXMC), also known as parallel tempering, is a parallelized version of the simulated tempering. In this algorithm, several copies of a system with different temperature, called as replicas, will be simulated in parallel. Then, with some interval of steps, each replica exchanges temperature with another one according to the (detailed) balance condition. As the simulated tempering does, RXMC can observe the details of a valley and escape from it by cooling and heating. Moreover, because each temperature is assigned to just one replica, the temperature distribution will not be biased. Using more replicas narrows the temperature interval, and increases the acceptance ratio of the temperature exchange. This is why this algorithm suits for the massively parallel calculation.

It is recommended that users perform `minsearch` optimization starting from the result of `exchange`, because the RXMC result has uncertainty due to temperature.

footnote

CHAPTER

SEVEN

TUTORIALS

The direct problem solver, `sim_trhepd_rheed`, is based on the Reflection-High-Energy Electron Diffraction (RHEED, TRHEPD) analysis software developed by Prof. Takashi Hanada at Tohoku University. In TRHEPD, when atomic coordinates are given, diffraction data is given as a simulation result. Therefore, we are dealing with the direct problem from atomic coordinates to diffraction data. On the other hand, in many cases, diffraction data is given experimentally, and the atomic coordinates are required to reproduce the experimental data. These are inverse problems to the above direct problems.

In 2DMAT, the algorithms for solving the inverse problem can be selected as following algorithms:

- `minsearch`

  Estimating plausible atomic coordinates using the Nealder-Mead method.

- `mapper_mpi`

  Estimate plausible atomic coordinates by searching the entire search grid for a given parameter.

- `bayes`

  Estimate plausible atomic coordinates using Bayesian optimization.

- `exchange`

  Sampling plausible atomic coordinates using a replica exchange Monte Carlo method.

In this tutorial, we will first introduce how to run the sequential problem program, and then how to run `minsearch` , `mapper_mpi`, `bayes`, and `exchange` .

7.1 TRHEPD Direct Problem Solver

As one of the forward problem solvers, 2DMAT provides a wrapper for the program sim-trhepd-rheed , which calculates the intensity of reflection fast (positron) electron diffraction (RHEED, TRHEPD) (A. Ichimiya, Jpn. J. Appl. Phys. 22, 176 (1983); 24, 1365 (1985)). In this tutorial, we will install and test sim-trhepd-rheed (for details, see the official web page for sim-trhepd-rheed).

7.1.1 Download and Install

First, in the tutorial, we assume that you are at the location where the `2DMAT` folder is located.

```
$ ls -d 2DMAT
2DMAT/
```

Get the source codes from the sim-trhepd-rheed repository on GitHub and build it.

```
git clone http://github.com/sim-trhepd-rheed/sim-trhepd-rheed
cd sim-trhepd-rheed/src
make
```

If make is successful, `bulk.exe` and `surf.exe` will be created.

7.1.2 Calculation execution

In sim-trhepd-rheed, the bulk part of the surface structure is first calculated with `bulk.exe`. Then, using the results of the `bulk.exe` calculation (the `bulkP.b` file), the surface portion of the `surf.exe` surface structure is calculated.

In this tutorial, we will actually try to do the TRHEPD calculation. The sample input files are located in `sample/sim-trhepd-rheed` in 2DMAT. First, copy this folder to a suitable working folder `work`.

```
cd ../../
cp -r 2DMAT/sample/sim-trhepd-rheed work
cd work
```

Next, copy `bulk.exe` and `surf.exe` to `work`.

```
cp ../sim-trhepd-rheed/src/bulk.exe .
cp ../sim-trhepd-rheed/src/surf.exe .
```

Execute `bulk.exe`.

```
./bulk.exe
```

Then, the bulk file `bulkP.b` will be generated with the following output.

```
0:electron 1:positron ?
P
input-filename (end=e) ? :
bulk.txt
output-filename :
bulkP.b
```

Next, execute `surf.exe`.

```
./surf.exe
```

Then, the following standard output will be seen.

```
bulk-filename (end=e) ? :
bulkP.b
structure-filename (end=e) ? :
surf.txt
output-filename :
```

(continues on next page)

(continued from previous page)

```
surf-bulkP.md
surf-bulkP.s
```

After execution, the files `surf-bulkP.md`, `surf-bulkP.s` and `SURFYYYYMMDD-HHMMSSlog.txt` will be generated. (YYYYMMDD and HHMMSS are numbers corresponding to the execution date and time).

7.1.3 Visualization of calculation result

The contents of `surf-bulkP.s` are shown as follow:

```
#azimuths,g-angles,beams
1 56 13
#ih,ik
6 0 5 0 4 0 3 0 2 0 1 0 0 0 -1 0 -2 0 -3 0 -4 0 -5 0 -6 0
0.5000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.
↪1595E-01, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.6000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.
↪1870E-01, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.7000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.
↪2121E-01, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.8000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.2171E-02, 0.
↪1927E-01, 0.2171E-02, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.9000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.4397E-02, 0.
↪1700E-01, 0.4397E-02, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.1000E+01, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.6326E-02, 0.
↪1495E-01, 0.6326E-02, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
...
```

From the above file, create a rocking curve from the angle on the vertical axis (first column of data after row 5) and the intensity of the (0,0) peak (eighth column of data after row 5). You can use Gnuplot or other graphing software, but here we use the program `plot_bulkP.py` in the `2DMAT/script` folder. Run it as follows.

```
python3 ../2DMAT/script/plot_bulkP.py
```

The following `plot_bulkP.png` will be created.

We will convolute and normalize the diffraction intensity data of the 00 peaks. Prepare `surf-bulkP.s` and run `make_convolution.py`.

```
python3 ../2DMAT/script/make_convolution.py
```

When executed, the following file `convolution.txt` will be created.

```
0.500000 0.010818010
0.600000 0.013986716
0.700000 0.016119093
0.800000 0.017039022
0.900000 0.017084666
  ... skipped ...
5.600000 0.000728539
5.700000 0.000530758
5.800000 0.000412908
5.900000 0.000341740
6.000000 0.000277553
```

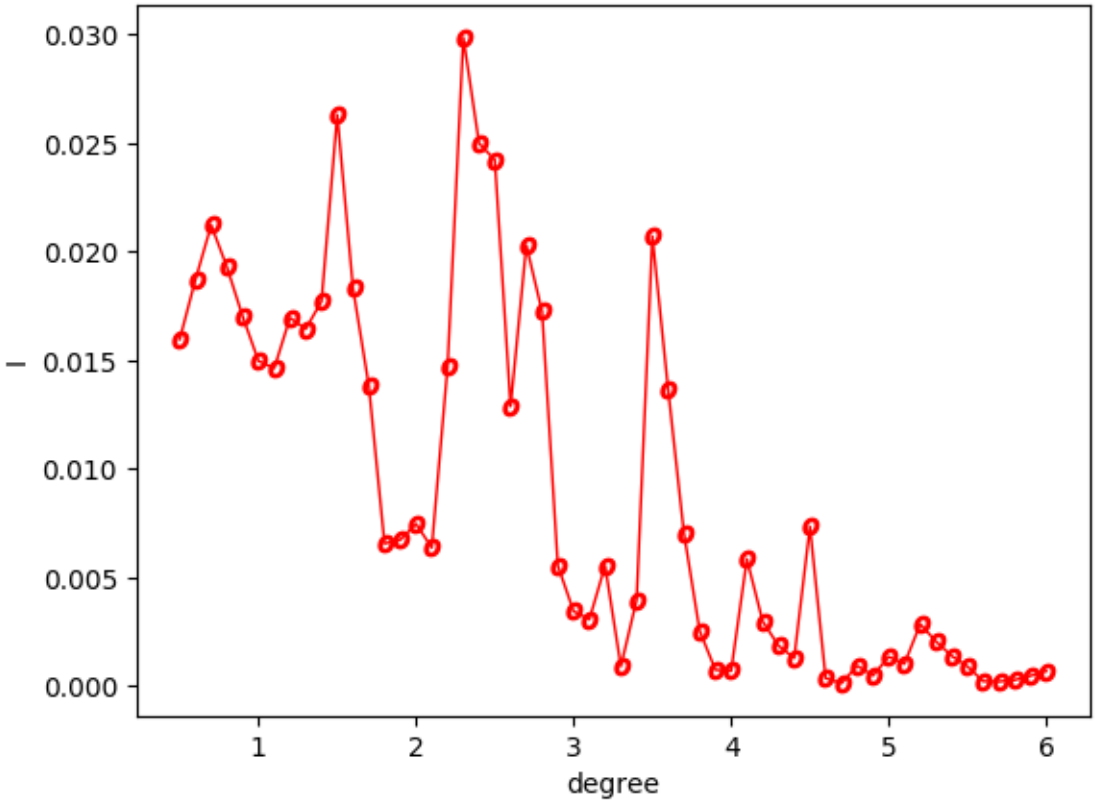

Fig. 7.1: Rocking curve of Si(001)-2x1 surface.

The first column is the viewing angle, and the second column is the normalized 00-peak diffraction intensity data written in `surf-bulkP.s` with a convolution of half-width 0.5.

7.2 Optimization by Nelder-Mead method

In this section, we will explain how to calculate the inverse problem of analyzing atomic coordinates from diffraction data using the Nelder-Mead method. The specific calculation procedure is as follows.

0. Preparation of the reference file

   Prepare the reference file to be matched (in this tutorial, it corresponds to `experiment.txt` described below).

1. Perform calculations on the bulk part of the surface structure.

   Copy `bulk.exe` to `sample/py2dmat/minsearch` and run the calculation.

2. Run the main program

   Run the calculation using `src/py2dmat_main.py` to estimate the atomic coordinates.

In the main program, the Nelder-Mead method (using scipy.optimize.fmin) is used.) to find the parameter that minimizes the deviation (R-value) between the intensity obtained using the solver (in this case `surf.exe`) and the intensity listed in the reference file (`experiment.txt`).

7.2.1 Location of the sample files

The sample files are located in `sample/py2dmat/minsearch`. The following files are stored in the folder.

- `bulk.txt`

  Input file of `bulk.exe`.

- `experiment.txt` , `template.txt`

  Reference file to proceed with calculations in the main program.

- `ref.txt`

  A file containing the answers you want to seek in this tutorial.

- `input.toml`

  Input file of the main program.

- `prepare.sh` , `do.sh`

  Script prepared for doing all calculation of this tutorial

The following sections describe these files and then show the actual calculation results.

7.2.2 The reference file

The `template.txt` file is almost the same format as the input file for `surf.exe`. The parameters to be run (such as the atomic coordinates you want to find) are rewritten as `value_*` or some other appropriate string. The following is the content of `template.txt`.

```
2                                   ,NELMS,  -------- Ge(001)-c4x2
32,1.0,0.1                          ,Ge Z,da1,sap
0.6,0.6,0.6                         ,BH(I),BK(I),BZ(I)
32,1.0,0.1                          ,Ge Z,da1,sap
0.4,0.4,0.4                         ,BH(I),BK(I),BZ(I)
9,4,0,0,2, 2.0,-0.5,0.5              ,NSGS,msa,msb,nsa,nsb,dthick,DXS,DYS
8                                   ,NATM
1, 1.0, 1.34502591  1       value_01   ,IELM(I),ocr(I),X(I),Y(I),Z(I)
1, 1.0, 0.752457792 1       value_02
2, 1.0, 1.480003343 1.465005851     value_03
2, 1.0, 2   1.497500418     2.281675
2, 1.0, 1   1.5     1.991675
2, 1.0, 0   1       0.847225
2, 1.0, 2   1       0.807225
2, 1.0, 1.009998328 1       0.597225
1,1                                 ,(WDOM,I=1,NDOM)
```

In this input file, `value_01`, `value_02`, and `value_03` are used. In the sample folder, there is a reference file `ref.txt` to know if the atomic positions are estimated correctly. The contents of this file are

```
fx = 7.382680568652868e-06
z1 = 5.230524973874179
z2 = 4.370622919269477
z3 = 3.5961444501081647
```

`value_0x` corresponds to `z_x` (x=1, 2, 3). `fx` is the optimal value of the objective function. The `experiment.txt` is a file that is used as a reference in the main program, and is equivalent to `convolution.txt`, which is calculated by putting the parameters in `ref.txt` into `template.txt` and following the same procedure as in the tutorial on direct problems. (Note that the input files for `bulk.exe` and `suft.exe` are different from those in the sequential problem tutorial.)

7.2.3 Input file

In this section, we will prepare the input file `input.toml` for the main program. The details of `input.toml` can be found in the input file. This section describes the contents of `input.toml` in the sample file.

```
[base]
dimension = 3

[solver]
name = "sim-trhepd-rheed"

[solver.config]
calculated_first_line = 5
calculated_last_line = 74
row_number = 2

[solver.param]
string_list = ["value_01", "value_02", "value_03" ]
degree_max = 7.0
```

(continues on next page)

(continued from previous page)

```
[solver.reference]
path = "experiment.txt"
first = 1
last = 70

[algorithm]
name = "minsearch"
label_list = ["z1", "z2", "z3"]

[algorithm.param]
min_list = [0.0, 0.0, 0.0]
max_list = [10.0, 10.0, 10.0]
initial_list = [5.25, 4.25, 3.50]
```

First, `[base]` section is explained.

- The `dimension` is the number of variables to be optimized, in this case `3` since we are optimizing three variables as described in `template.txt`.

The `[solver]` section specifies the solver to be used inside the main program and its settings.

- The `name` is the name of the solver you want to use, which in this tutorial is `sim-trhepd-rheed`, since we will be using it for our analysis.

The solver can be configured in the subsections `[solver.config]`, `[solver.param]`, and `[solver.reference]`.

The `[solver.config]` section specifies options for reading the output file produced by the main program's internal call, `surf.exe`.

- The `calculated_first_line` specifies the first line to read from the output file.
- The `calculated_last_line` specifies the last line of the output file to be read.
- The `row_number` specifies the number of columns in the output file to read.

The `[solver.param]` section specifies options for reading the output file produced by the main program's internal call, `surf.exe`.

- The `string_list` is a list of variable names to be read in `template.txt`.
- `degree_max` specifies the maximum angle in degrees.

The `[solver.reference]` section specifies the location of the experimental data and the range to read.

- The `path` specifies the path where the experimental data is located.
- The `first` specifies the first line of the experimental data file to read.
- The `end` specifies the last line of the experimental data file to read.

The `[algorithm]` section specifies the algorithm to use and its settings.

- The `name` is the name of the algorithm you want to use, in this tutorial we will use `minsearch` since we will be using the Nelder-Mead method.
- The `label_list` is a list of label names to be added to the output of `value_0x` (x=1,2,3).

The `[algorithm.param]` section specifies the range of parameters to search and their initial values.

- The `min_list` and `max_list` specify the minimum and maximum values of the search range, respectively.
- The `initial_list` specifies the initial values.

Other parameters, such as convergence judgments used in the Nelder-Mead method, can be done in the `[algorithm]` section, although they are omitted here because the default values are used. See the input file chapter for details.

7.2.4 Calculation execution

First, move to the folder where the sample files are located (we will assume that you are directly under the directory where you downloaded this software).

```
cd sample/py2dmat/minsearch
```

Copy `bulk.exe` and `surf.exe`.

```
cp ../../../../sim-trhepd-rheed/src/TRHEPD/bulk.exe .
cp ../../../../sim-trhepd-rheed/src/TRHEPD/surf.exe .
```

First, run `bulk.exe` to create `bulkP.b`.

```
./bulk.exe
```

After that, run the main program (the computation time takes only a few seconds on a normal PC).

```
python3 ../../../src/py2dmat_main.py input.toml | tee log.txt
```

Then, the standard output will be seen as follows.

```
Read experiment.txt
z1 =  5.25000
z2 =  4.25000
z3 =  3.50000
[' 5.25000', ' 4.25000', ' 3.50000']
PASS : degree in lastline = 7.0
PASS : len(calculated_list) 70 == len(convolution_I_calculated_list)70
R-factor = 0.015199251773721183
z1 =  5.50000
z2 =  4.25000
z3 =  3.50000
[' 5.50000', ' 4.25000', ' 3.50000']
PASS : degree in lastline = 7.0
PASS : len(calculated_list) 70 == len(convolution_I_calculated_list)70
R-factor = 0.04380131351780189
z1 =  5.25000
z2 =  4.50000
z3 =  3.50000
[' 5.25000', ' 4.50000', ' 3.50000']
...
```

The `z1`, `z2`, and `z3` are the candidate parameters at each step and the `R-factor` at that time. The results of each step are also output to the folder `Logxxxxx` (where xxxxx is the number of steps). The final estimated parameters will be output to `res.dat`. In the current case, the following result is obtained:

```
z1 = 5.230524973874179
z2 = 4.370622919269477
z3 = 3.5961444501081647
```

You can see that we get the same value as the correct answer data `ref.txt`. Note that `do.sh` is available as a script for batch calculation. In `do.sh`, it also compares the difference between `res.txt` and `ref.txt`. Here is what it does, without further explanation.

```
sh ./prepare.sh

./bulk.exe

time python3 ../../../src/py2dmat_main.py input.toml | tee log.txt

echo diff res.txt ref.txt
res=0
diff res.txt ref.txt || res=$?
if [ $res -eq 0 ]; then
  echo Test PASS
  true
else
  echo Test FAILED: res.txt and ref.txt differ
  false
fi
```

7.2.5 Visualization of calculation results

The data of the rocking curve at each step is stored in `Logxxxxx` (where xxxx is the number of steps) as `RockingCurve.txt`. A tool `draw_RC_double.py` is provided to visualize this data. In this section, we will use this tool to visualize the results.

```
cp 0/Log00000001/RockingCurve.txt RockingCurve_ini.txt
cp 0/Log00000017/RockingCurve.txt RockingCurve_con.txt
cp ../../../script/draw_RC_double.py .
python draw_RC_double.py
```

Running the above will output `RC_double_minsearch.png`.

From the figure, we can see that the last step agrees with the experimental one.

7.3 Grid search

In this section, we will explain how to perform a grid-type search and analyze atomic coordinates from diffraction data. The grid type search is compatible with MPI. The specific calculation procedure is the same as for `minsearch`. However, it is necessary to prepare the data `MeshData.txt` to give the search grid in advance.

7.3.1 Location of the sample files

The sample files are located in `sample/py2dmat/mapper`. The following files are stored in the folder

- `bulk.txt`

  Input file of `bulk.exe`

- `experiment.txt` , `template.txt`

  Reference file to proceed with calculations in the main program.

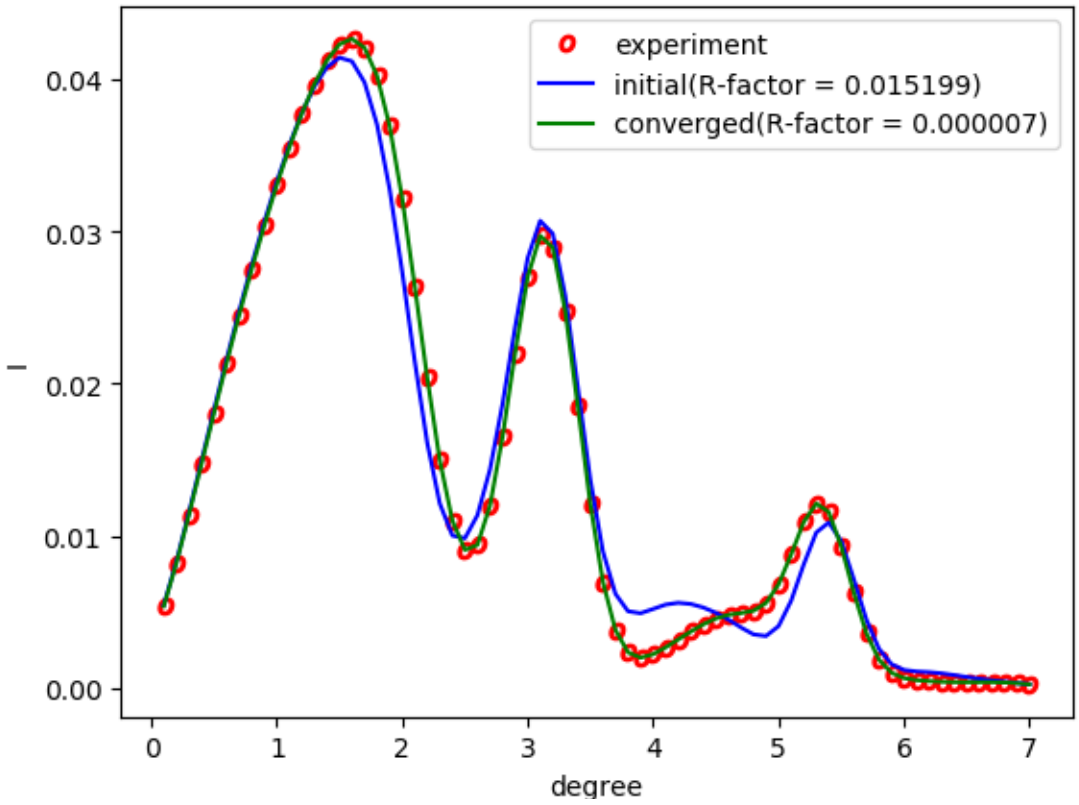

Fig. 7.2: Analysis using the Nelder-Mead method. The red circle represents the experimental value, the blue line represents the first step, and the green line represents the rocking curve obtained at the last step.

- `ref_ColorMap.txt`

  A file to check if the calculation was performed correctly (the answer to `ColorMap.txt` obtained by doing this tutorial).

- `input.toml`

  Input file of the main program.

- `prepare.sh` , `do.sh`

  Script prepared for bulk calculation of this tutorial.

Below, we will describe these files and then show the actual calculation results.

7.3.2 Reference file

The `template.txt` and `experiment.txt` are the same as in the previous tutorial (Nealder-Mead optimization). However, to reduce the computation time, the value is fixed to `3.5` instead of `value_03`, and the grid is searched in 2D. The actual grid to be searched is given in `MeshData.txt`. In the sample, the contents of `MeshData.txt` are as follows.

```
1 6.000000 6.000000
2 6.000000 5.750000
3 6.000000 5.500000
4 6.000000 5.250000
5 6.000000 5.000000
6 6.000000 4.750000
7 6.000000 4.500000
8 6.000000 4.250000
9 6.000000 4.000000
...
```

The first column is the serial number, and the second and subsequent columns are the values of `value_0` , `value_1` that go into `template.txt`, in that order.

7.3.3 Input file

This section describes the input file for the main program, `input.toml`. The details of `input.toml` can be found in the input file. The following is the content of `input.toml` in the sample file.

```
[base]
dimension = 2

[solver]
name = "sim-trhepd-rheed"

[solver.config]
calculated_first_line = 5
calculated_last_line = 74
row_number = 2

[solver.param]
string_list = ["value_01", "value_02" ]
degree_max = 7.0

[solver.reference]
```

(continues on next page)

(continued from previous page)

```
path = "experiment.txt"
first = 1
last = 70

[algorithm]
name = "mapper"
label_list = ["z1", "z2"]
```

First, `[base]` section is explained.

- The `dimension` is the number of variables to be optimized, in this case `2` since we are optimizing two variables as described in `template.txt`.

The `[solver]` section specifies the solver to be used inside the main program and its settings.

- The `name` is the name of the solver you want to use, which in this tutorial is `sim-trhepd-rheed`, since we will be using it for our analysis.

The solver can be configured in the subsections `[solver.config]`, `[solver.param]`, and `[solver.reference]`.

The `[solver.config]` section specifies options for reading the output file produced by the main program's internal call, `surf.exe`.

- The `calculated_first_line` specifies the first line to read from the output file.
- The `calculated_last_line` specifies the last line of the output file to be read.
- The `row_number` specifies the number of columns in the output file to read.

The `[solver.param]` section specifies options for reading the output file produced by the main program's internal call, `surf.exe`.

- The `string_list` is a list of variable names to be read in `template.txt`.
- `degree_max` specifies the maximum angle in degrees.

The `[solver.reference]` section specifies the location of the experimental data and the range to read.

- The `path` specifies the path where the experimental data is located.
- The `first` specifies the first line of the experimental data file to read.
- The `end` specifies the last line of the experimental data file to read.

The `[algorithm]` section specifies the algorithm to use and its settings.

- The `name` is the name of the algorithm you want to use, in this tutorial we will use `mapper` since we will be using grid-search method.
- The `label_list` is a list of label names to be attached to the output `value_0x` (x=1,2).

For details on other parameters that can be specified in the input file, please see the Input File chapter.

7.3.4 Calculation execution

First, move to the folder where the sample files are located (we will assume that you are directly under the directory where you downloaded this software).

```
cd sample/py2dmat/minsearch
```

Copy `bulk.exe` and `surf.exe`.

```
cp ../../../../sim-trhepd-rheed/src/TRHEPD/bulk.exe .
cp ../../../../sim-trhepd-rheed/src/TRHEPD/surf.exe .
```

First, run `bulk.exe` to create `bulkP.b`.

```
./bulk.exe
```

After that, run the main program (the computation time takes only a few seconds on a normal PC).

```
mpiexec -np 2 python3 ../../../src/py2dmat_main.py input.toml | tee log.txt
```

Here, the calculation using MPI parallel with 2 processes will be done. When executed, a folder for each rank will be created, and a subfolder `Log%%%%%` (where `%%%%%` is the grid id) will be created under it. (The grid id is assigned to the number in `MeshData.txt`). The standard output will be seen like this.

```
Iteration : 1/33
Read experiment.txt
mesh before: [1.0, 6.0, 6.0]
z1 =  6.00000
z2 =  6.00000
[' 6.00000', ' 6.00000']
PASS : degree in lastline = 7.0
PASS : len(calculated_list) 70 == len(convolution_I_calculated_list)70
R-factor = 0.04785241875354398
...
```

The `z1` and `z2` are the candidate parameters for each mesh and the `R-factor` at that time. Finally, the `R-factor` calculated for all the points on the grid will be output to `ColorMap.txt`. In this case, the following results will be obtained.

```
6.000000 6.000000 0.047852
6.000000 5.750000 0.055011
6.000000 5.500000 0.053190
6.000000 5.250000 0.038905
6.000000 5.000000 0.047674
6.000000 4.750000 0.065919
6.000000 4.500000 0.053675
6.000000 4.250000 0.061261
6.000000 4.000000 0.069351
6.000000 3.750000 0.071868
6.000000 3.500000 0.072739
...
```

The first and second columns will contain the values of `value_01` and `value_02`, and the third column will contain the `R-factor`. Note that `do.sh` is available as a script for batch calculation. In `do.sh`, it also compares the difference between `res.txt` and `ref.txt`. Here is what it does, without further explanation.

```
sh prepare.sh

./bulk.exe

time mpiexec -np 2 python3 ../../../src/py2dmat_main.py input.toml

echo diff ColorMap.txt ref_ColorMap.txt
res=0
diff ColorMap.txt ref_ColorMap.txt || res=$?
if [ $res -eq 0 ]; then
  echo TEST PASS
  true
else
  echo TEST FAILED: ColorMap.txt and ref_ColorMap.txt differ
  false
fi
```

7.3.5 Visualization of calculation results

By seeing `ColorMap.txt`, we can estimate the region where the small parameters of `R-factor` are located. In this case, the following command will create a two-dimensional parameter space diagram `ColorMapFig.png`.

```
python3 plot_colormap_2d.py
```

Looking at the generated figure, we can see that it has a minimum value around (5.25, 4.25).

`RockingCurve.txt` is stored in each subfolder. By using it, you can compare the results with the experimental values following the procedure in the previous tutorial.

7.4 Optimization by Bayesian Optimization

This tutorial subscribes how to estimate atomic positions from the experimental diffraction data by using Bayesian optimization (BO). 2DMAT uses PHYSBO for BO.

7.4.1 Sample files

Sample files are available from `sample/py2dmat/bayes` . This directory includes the following files:

- `bulk.txt`

  The input file of `bulk.exe`

- `experiment.txt` , `template.txt`

  Reference files for the main program

- `ref_BayesData.txt`

  Solution file for checking whether the calucation successes or not

- `input.toml`

  The input file of py2dmat

- `prepare.sh` , `do.sh`

  Script files for running this tutorial

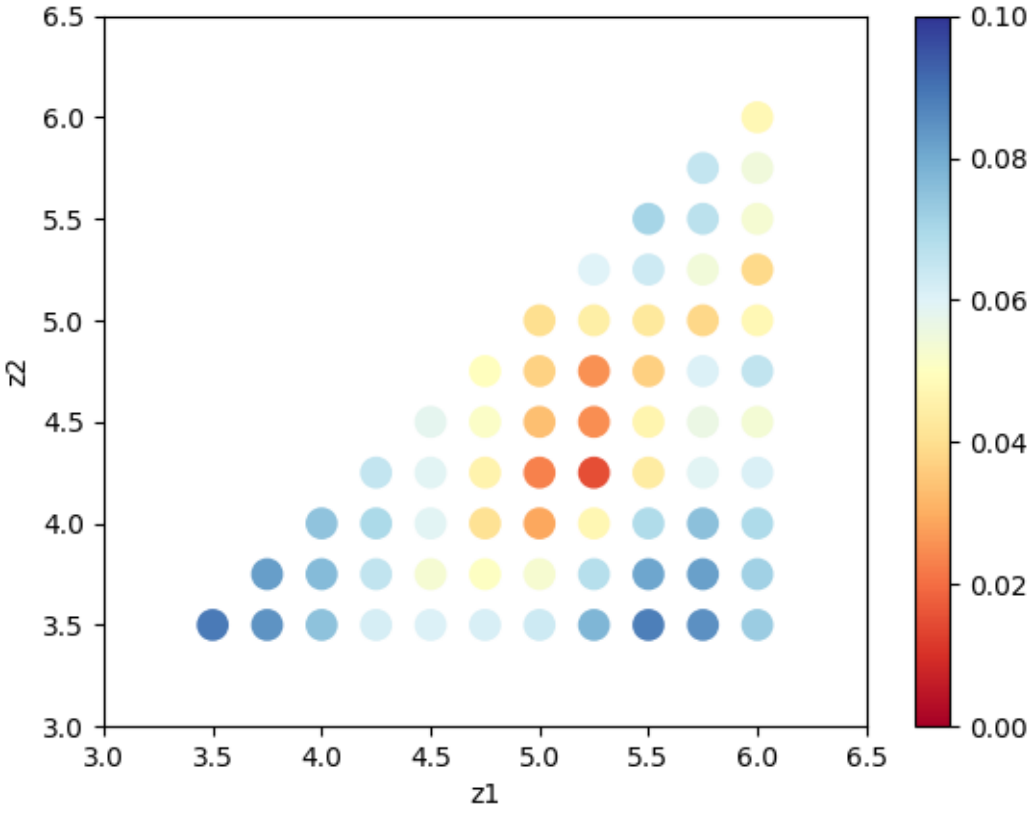

Fig. 7.3: `R-factor` on a two-dimensional parameter space.

In the following, we will subscribe these files and then show the result.

7.4.2 Reference files

This tutorial uses `template.txt` , `experiment.txt` similar to the previous one (`minsearch`). Only difference is that in this tutorial the third parameter `value_03` is fixed to `3.5` in order to speed up the calculation. The parameter space to be explored is given by `MeshData.txt`.

```
1 6.000000 6.000000
2 6.000000 5.750000
3 6.000000 5.500000
4 6.000000 5.250000
5 6.000000 5.000000
6 6.000000 4.750000
7 6.000000 4.500000
8 6.000000 4.250000
9 6.000000 4.000000
...
```

The first column is the index of the point and the remaining ones are the coodinates, `value_0` and `value_1` in the `template.txt`.

7.4.3 Input files

This subsection describes the input file. For details, see *the manual of bayes*. `input.toml` in the sample directory is shown as the following

```
[base]
dimension = 2

[solver]
name = "sim-trhepd-rheed"

[solver.config]
calculated_first_line = 5
calculated_last_line = 74
row_number = 2

[solver.param]
string_list = ["value_01", "value_02" ]
degree_max = 7.0

[solver.reference]
path = "experiment.txt"
first = 1
last = 70

[algorithm]
name = "bayes"
label_list = ["z1", "z2"]

[algorithm.param]
mesh_path = "MeshData.txt"

[algorithm.bayes]
```

(continues on next page)

(continued from previous page)

```
random_max_num_probes = 5
bayes_max_num_probes = 20
```

- The [base] section describes the settings for a whole calculation.
  - dimension is the number of variables you want to optimize. In this case, specify 2 because it optimizes two variables.
- The [solver] section specifies the solver to use inside the main program and its settings.
  - See the minsearch tutorial.
- The [algorithm] section sets the algorithm to use and its settings.
  - name is the name of the algorithm you want to use, and in this tutorial we will do a Bayesian optimization analysis, so specify bayes.
  - label_list is a list of label names to be given when outputting the value of value_0x (x = 1,2).
  - The [algorithm.bayes] section sets the parameters for Bayesian optimization.
    - random_max_num_probes specifies the number of random searches before Bayesian optimization.
    - bayes_max_num_probes specifies the number of Bayesian searches.

For details on other parameters that can be specified in the input file, see the chapter on input files of bayes.

7.4.4 Calculation

First, move to the folder where the sample file is located (hereinafter, it is assumed that you are the root directory of 2DMAT).

```
cd sample/py2dmat/bayes
```

Copy bulk.exe and surf.exe as the tutorial for the direct problem.

```
cp ../../../../sim-trhepd-rheed/src/TRHEPD/bulk.exe .
cp ../../../../sim-trhepd-rheed/src/TRHEPD/surf.exe .
```

Execute bulk.exe to generate bulkP.b .

```
./bulk.exe
```

Then, run the main program (it takes a few secondes)

```
python3 ../../../src/py2dmat_main.py input.toml | tee log.txt
```

This makes a directory with the name of 0 . The following standard output will be shown:

```
#parameter
 random_max_num_probes = 5
 bayes_max_num_probes = 20
 score = TS
 interval = 5
 num_rand_basis = 5000
 Read MeshData.txt
 value_01 =  4.75000
```

(continues on next page)

(continued from previous page)

```
value_02 =  4.50000
WARNING : degree in lastline = 7.0, but 6.0 expected
PASS : len(calculated_list) 70 == len(convolution_I_calculated_list)70
R-factor = 0.05141906746102885
0001-th step: f(x) = -0.051419 (action=46)
   current best f(x) = -0.051419 (best action=46)

value_01 =  6.00000
value_02 =  4.75000
...
```

A list of hyperparameters, followed by candidate parameters at each step and the corresponding `R-factor` multiplied by -1, are shown first. It also outputs the grid index (`action`) and `f(x)` with the best `R-factor` at that time. Under the directory `0`, subdirectories with the name is the grid id are created, like `Log%%%%%` (`%%%%%` is the grid id), and the solver output for each grid is saved. (The first column in `MeshData.txt` will be assigned as the id of the grid). The final estimated parameters are output to `BayesData.txt`.

In this case, `BayesData.txt` can be seen as the following

```
#step z1 z2 R-factor z1_action z2_action R-factor_action
0 4.75 4.5 0.05141906746102885 4.75 4.5 0.05141906746102885
1 4.75 4.5 0.05141906746102885 6.0 4.75 0.06591878368102033
2 5.5 4.25 0.04380131351780189 5.5 4.25 0.04380131351780189
3 5.0 4.25 0.02312528177606794 5.0 4.25 0.02312528177606794
4 5.0 4.25 0.02312528177606794 6.0 5.75 0.05501069117756031
5 5.0 4.25 0.02312528177606794 5.0 4.75 0.037158316568603085
6 5.0 4.25 0.02312528177606794 5.75 4.75 0.060611944378678 95
7 5.0 4.25 0.02312528177606794 4.25 3.5 0.062098618649988294
8 5.0 4.25 0.02312528177606794 6.0 6.0 0.04785241875354398
9 5.0 4.25 0.02312528177606794 4.5 4.0 0.05912332368374844
10 5.0 4.25 0.02312528177606794 4.75 4.25 0.04646333628698967
11 5.0 4.25 0.02312528177606794 5.5 4.5 0.0466682914488051
12 5.0 4.25 0.02312528177606794 5.0 4.5 0.033464998538380517
13 5.25 4.25 0.015199251773721183 5.25 4.25 0.015199251773721183
14 5.25 4.25 0.015199251773721183 5.25 4.0 0.04752465769047 07
...
```

The first column contains the number of steps, and the second, third, and fourth columns contain `` value_01``, `` value_02``, and `` R-factor``, which give the highest score at that time. This is followed by the candidate `value_01`, `value_02` and `R-factor` for that step. In this case, you can see that the correct solution is obtained at the 13th step.

In addition, `do.sh` is prepared as a script for batch calculation. `do.sh` also checks the difference between `BayesData.dat` and `ref_BayesData.dat`. I will omit the explanation below, but I will post the contents.

```
sh prepare.sh

./bulk.exe

time python3 ../../../src/py2dmat_main.py input.toml

echo diff BayesData.txt ref_BayesData.txt
res=0
diff BayesData.txt ref_BayesData.txt || res=$?
if [ $res -eq 0 ]; then
  echo TEST PASS
  true
```

(continues on next page)

(continued from previous page)

```
else
  echo TEST FAILED: BayesData.txt.txt and ref_BayesData.txt.txt differ
  false
fi
```

7.4.5 Visualization

You can see at what step the parameter gave the minimum score by looking at `BayesData.txt`. Since `RockingCurve.txt` is stored in a subfolder for each step, it is possible to compare it with the experimental value by following the procedure of :doc:`minsearch`.

7.5 Optimization by replica exchange Monte Carlo

This tutorial subscribes how to estimate atomic positions from the experimental diffraction data by using the replica exchange Monte Carlo method (RXMC).

7.5.1 Sample files

Sample files are available from `sample/py2dmat/bayes` . This directory includes the following files:

- `bulk.txt`

  The input file of `bulk.exe`

- `experiment.txt` , `template.txt`

  Reference files for the main program

- `ref.txt`

  Solution file for checking whether the calucation successes or not

- `input.toml`

  The input file of py2dmat

- `prepare.sh` , `do.sh`

  Script files for running this tutorial

In the following, we will subscribe these files and then show the result.

7.5.2 Reference files

This tutorial uses reference files, `template.txt` and `experiment.txt`, which are the same as the previous tutorial (*Optimization by Nelder-Mead method*) uses.

7.5.3 Input files

This subsection describes the input file. For details, see *the manual of bayes*. `input.toml` in the sample directory is shown as the following

```
[base]
dimension = 2

[algorithm]
name = "exchange"
label_list = ["z1", "z2"]
seed = 12345

[algorithm.param]
min_list = [3.0, 3.0]
max_list = [6.0, 6.0]

[algorithm.exchange]
numsteps = 1000
numsteps_exchange = 20
Tmin = 0.005
Tmax = 0.05
Tlogspace = true

[solver]
name = "sim-trhepd-rheed"

[solver.config]
calculated_first_line = 5
calculated_last_line = 74
row_number = 2

[solver.param]
string_list = ["value_01", "value_02" ]
degree_max = 7.0

[solver.reference]
path = "experiment.txt"
first = 1
last = 70
```

In the following, we will briefly describe this input file. For details, see the manual of *Replica exchange Monte Carlo exchange*.

- The `[base]` section describes the settings for a whole calculation.
  - `dimension` is the number of variables you want to optimize. In this case, specify `2` because it optimizes two variables.
- The `[solver]` section specifies the solver to use inside the main program and its settings.
  - See the minsearch tutorial.
- The `[algorithm]` section sets the algorithm to use and its settings.
  - `name` is the name of the algorithm you want to use, and in this tutorial we will use RXMC, so specify `exchange`.
  - `label_list` is a list of label names to be given when outputting the value of `value_0x` (x = 1,2).
  - `seed` is the seed that a pseudo-random number generator uses.

  - The `[algorithm.param]` section sets the parameter space to be explored.
    * `min_list` is a lower bound and `max_list` is an upper bound.
  - The `[algorithm.exchange]` section sets the parameters for RXMC.
    * `numstep` is the number of Monte Carlo steps.
    * `numsteps_exchange` is the number of interval steps between temperature exchanges.
    * `Tmin`, `Tmax` are the minimum and the maximum of temperature, respectively.
    * When `Tlogspace` is `true`, the temperature points are distributed uniformly in the logarithmic space.
- The `[solver]` section specifies the solver to use inside the main program and its settings.
  - See the *Optimization by Nelder-Mead method* tutorial.

7.5.4 Calculation

First, move to the folder where the sample file is located (hereinafter, it is assumed that you are the root directory of 2DMAT).

```
cd sample/py2dmat/bayes
```

Copy `bulk.exe` and `surf.exe` as the tutorial for the direct problem.

```
cp ../../../../sim-trhepd-rheed/src/TRHEPD/bulk.exe .
cp ../../../../sim-trhepd-rheed/src/TRHEPD/surf.exe .
```

Execute `bulk.exe` to generate `bulkP.b` .

```
./bulk.exe
```

Then, run the main program (it takes a few secondes)

```
mpiexec -np 4 python3 ../../../src/py2dmat_main.py input.toml | tee log.txt
```

Here, the calculation is performed using MPI parallel with 4 processes. (If you are using Open MPI and you request more processes than you can use, add the `--oversubscribed` option to the `mpiexec` command.)

When executed, a folder for each rank will be created, and a `trial.txt` file containing the parameters evaluated in each Monte Carlo step and the value of the objective function, and a `result.txt` file containing the parameters actually adopted will be created.

These files have the same format: the first column is the number of steps, the second is the temperature, the third column is the value of the objective function, and the fourth and subsequent columns are the parameters.

```
# step T fx x1 x2
0 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
1 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
2 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
3 0.004999999999999999 0.06273922648753057 4.330900869594549 4.311333132184154
```

In the case of the sim-trhepd-rheed solver, a subfolder `Log%%%%%` (`%%%%%` is the number of MC steps) is created under each working folder, and locking curve information etc. are recorded.

Finally, `best_result.txt` is filled with information about the parameter with the optimal objective function (R-factor), the rank from which it was obtained, and the Monte Carlo step.

```
nprocs = 4
rank = 2
step = 65
fx = 0.008233957976993406
x[0] = 4.221129370933539
x[1] = 5.139591716517661
```

In addition, `do.sh` is prepared as a script for batch calculation. `do.sh` also checks the difference between `best_result.txt` and `ref.txt`. I will omit the explanation below, but I will post the contents.

```
sh prepare.sh

./bulk.exe

time mpiexec --oversubscribe -np 4 python3 ../../../src/py2dmat_main.py input.toml

echo diff best_result.txt ref.txt
res=0
diff best_result.txt ref.txt || res=$?
if [ $res -eq 0 ]; then
  echo TEST PASS
  true
else
  echo TEST FAILED: best_result.txt and ref.txt differ
  false
fi
```

7.5.5 Post process

The `result.txt` in each rank folder records the data sampled by each replica, but the same replica holds samples at different temperatures because of the temperature exchanges. 2DMat provides a script, `script/separateT.py`, that rearranges the results of all replicas into samples by temperature.

```
python3 ../../../script/separateT.py
```

The data reorganized for each temperature point is written to `result_T%.txt` (`%` is the index of the temperature point). The first column is the step, the second column is the rank, the third column is the value of the objective function, and the fourth and subsequent columns are the parameters.

Example:

```
# T = 0.004999999999999999
# step rank fx x1 x2
0 0 0.07830821484593968 3.682008067401509 3.9502750191292586
1 0 0.07830821484593968 3.682008067401509 3.9502750191292586
2 0 0.07830821484593968 3.682008067401509 3.9502750191292586
```

7.5.6 Visualization

By illustrating `result_T.txt`, you can estimate regions where the parameters with small R-factor are. In this case, the figure `result.png` of the 2D parameter space is created by using the following command.

```
python3 plot_result_2d.py
```

Looking at the resulting diagram, we can see that the samples are concentrated near (5.25, 4.25) and (4.25, 5.25), and that the `R-factor` value is small there.

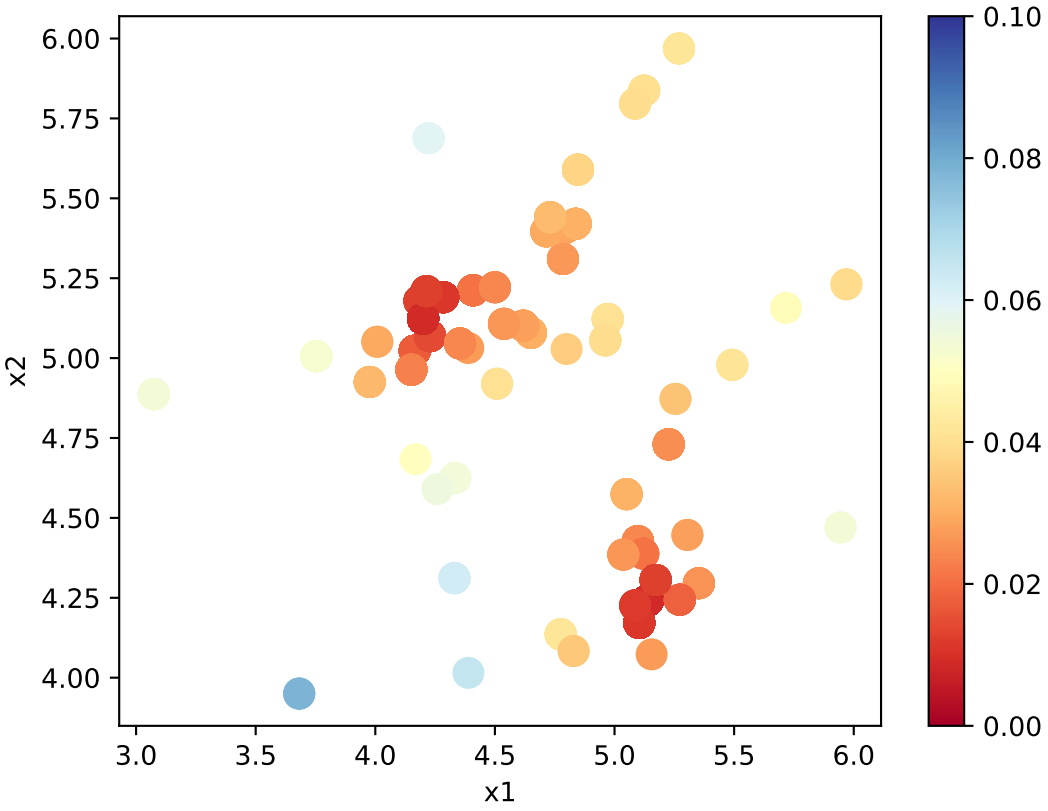

Fig. 7.4: Sampled parameters and `R-factor`. The horizontal axes is `value_01` and the vertical axes is `value_02` .

Also, `RockingCurve.txt` is stored in each subfolder. By using this, it is possible to compare with the experimental value according to the procedure of the previous tutorial.

CHAPTER

EIGHT

RELATED TOOLS

8.1 to_dft.py

This tool generates input data for Quantum Espresso (QE) , a first-principles electronic structure calculation software, from the atomic structures of (001) and (111) surface models of systems with Si isotetrahedral bond networks. This is used to validate the obtained structure and to obtain microscopic information such as the electronic state. In order to eliminate the influence of dangling bond-derived electrons from the opposite surface of interest, we use a technique called hydrogen termination, in which a hydrogen atom is placed at the position of the lowest dangling bond.

8.1.1 Prerequisites

- Python3 >= 3.6

The following packages are required:

- Atomic Simulation Environment(ASE) (>= 3.21.1)
- Numpy
- Scipy
- Matplotlib

8.1.2 Overview of this tool

The input file including the information such as the name of the structure file (XYZ format) and the lattice vector information to represent the two-dimensional periodic structure is read in, and the coordinates of the lowest layer and the next layer of atoms are extracted from the obtained coordinate data. The bottom layer atoms are removed, and H atoms are placed at the corresponding positions to create a model with the distance to the next layer atoms adjusted to a tetrahedral structure (for example, the distance to a silane molecule in the case of Si). The hydrogen-terminated model is saved in XYZ format, and a cif file and an input file for Quantum Espresso (QE) are also created. If you have QE installed, you can also run the calculation as is.

8.1.3 Tutorial

1. Prepare an XYZ file for reference.

In the following, we will use the file `surf_bulk_new111.xyz` in the folder `tool/todft/sample/111`. The contents of the file are as follows.

```
12
 surf.txt              / bulk.txt
 Si     1.219476    0.000000    4.264930
 Si     6.459844    0.000000    4.987850
 Si     1.800417    1.919830    3.404650
 Si     5.878903    1.919830    3.404650
 Si     3.839660    1.919830    2.155740
 Si     0.000000    1.919830    1.900440
 Si     3.839660    0.000000    0.743910
 Si     0.000000    0.000000    0.597210
 Si     1.919830    0.000000   -0.678750
 Si     5.759490    0.000000   -0.678750
 Si     1.919830    1.919830   -2.036250
 Si     5.759490    1.919830   -2.036250
```

2. Next, create an input file for setting the various parameters.

The file format of the input file is `toml`. The following section describes the contents of the input file using `input.toml` in the `tool/todft/sample/111` folder. The contents of the file are as follows.

```
[Main]
input_xyz_file = "surf_bulk_new111.xyz"
output_file_head = "surf_bulk_new111_ext"
[Main.param]
z_margin = 0.001
slab_margin = 10.0
r_SiH = 1.48 #angstrom
theta = 109.5 #H-Si-H angle in degree
[Main.lattice]
unit_vec = [[7.67932, 0.00000, 0.00000], [0.00000, 3.83966, 0.00000]]
[ASE]
solver_name = "qe"
kpts = [3,3,1]         # sampling k points (Monkhorst-Pack grid)
command = "mpirun -np 4 ./pw.x -in espresso.pwi > espresso.pwo"
[Solver]
[Solver.control]
calculation='bands' # 'scf','realx','bands',...
pseudo_dir='./'     # Pseudopotential directory
[Solver.system]
ecutwfc = 20.0         # Cut-off energy in Ry
nbands=33            # # of bands (only used in band structure calc
[Solver.pseudo]
Si = 'Si.pbe-mt_fhi.UPF'
H = 'H.pbe-mt_fhi.UPF'
```

The input file consists of three sections: `Main`, `ASE`, and `Solver`. Below is a brief description of the variables for each section.

`Main` section

This section contains settings related to the parameters required for hydrogen termination.

- `input_xyz_file`

  Format: string

  Description: Name of the xyz file to input

- `output_file_head`

  Format: string

  Description: Header for output files (xyz and cif files)

`Main.Param` section

- `z_margin`

  Format: float

  Description: Margin used to extract the lowest and second-to-last atoms. For example, if the z-coordinate of the atom in the bottom layer is `z_min`, the atoms in `z_min - z_margin <= z <= z_min + z_margin` will be extracted.

- `slab_margin`

  Format: float

  Description: Margin for tuning the size of the slab. If the z-coordinates of the atoms in the bottom and top layers are `z_min` , `z_max`, then the slab size is given by `z_max-z_min+slab_margin`.

- `r_SiH`

  Format: float

  Description: The distance (in Å) between a vertex (e.g. Si) and H of a tetrahedral structure.

- `theta`

  Format: float

  Description: The angle between the vertex and H of the tetrahedral structure (e.g. Si-H-Si).

`Main.lattice` section

- `unit_vec`

  Format: list

  Description: Specify a unit vector that forms a 2D plane (ex. `unit_vec = [[7.67932, 0.00000, 0.00000], [0.00000, 3.83966, 0.00000]]`).

`ASE` section

This section specifies parameters related to `ASE`.

- `solver_name`

  Format: string

  Description: The name of the solver. Currently, only `qe` is given.

- `kpts`

  Format: list

  Description: Specify the k-points to be sampled (Monkhorst-Pack grid).

- `command`

  Format: string

  Description: Set the command used to run the solver.

`Solver` section

In this section, parameters related to `Solver` are specified. You will need to specify this if you want to perform first-principles calculations directly using ASE. Basically, the configuration is the same as the one specified in the input file of each solver. For example, in the case of QE, `Solver.control` contains the parameters to be set in the `control` section of QE.

3. Execute the following command.

```
python3 to_dft.py input.toml
```

After finishing calculations, the following files are generated:

- `surf_bulk_new111_ext.xyz`
- `surf_bulk_new111_ext.cif`
- `espresso.pwi`

If the path to the QE and pseudopotential is set in the input file, the first-principle calculation will be performed as is. If not, the ab initio calculation will not be performed and you will get the message `Calculation of get_potential_energy is not normally finished.` at the end, but the above file will still be output.

The following is a description of the output file.

- `surf_bulk_new111_ext.xyz`

The output is the result of the replacement of the lowest level atom with H and the addition of H to form a tetrahedral structure. The actual output is as follows.

```
14
Lattice="7.67932 0.0 0.0  0.0 3.83966 0.0  0.0 0.0 17.0241"␣
↪Properties=species:S:1:pos:R:3 pbc="T T T"
Si  1.219476  0.000000  4.264930
Si  6.459844  0.000000  4.987850
Si  1.800417  1.919830  3.404650
Si  5.878903  1.919830  3.404650
Si  3.839660  1.919830  2.155740
Si  0.000000  1.919830  1.900440
Si  3.839660  0.000000  0.743910
```

(continues on next page)

(continued from previous page)

```
Si  0.000000  0.000000  0.597210
Si  1.919830  0.000000  -0.678750
Si  5.759490  0.000000  -0.678750
H  1.919830  -1.208630  -1.532925
H  1.919830  1.208630  -1.532925
H  5.759490  -1.208630  -1.532925
H  5.759490  1.208630  -1.532925
```

This file can be read by appropriate visualization software as ordinary XYZFormat coordinate data, but the lattice vector information of the periodic structure is written in the place where comments are usually written. You can also copy the data of "element name + 3D coordinate" from the third line of the output file to the input file of QE.

`espresso.pwi` is the input file for QE's scf calculation, and structural optimization and band calculation can be done by modifying this file accordingly. For details, please refer to the QE online manual .

CHAPTER

NINE

(FOR DEVELOPERS) USER-DEFINED ALGORITHM AND SOLVER

`py2dmat` solves the reverse problem by combination of `Solver` for the direct problem and `Algorithm` for the optimization problem. Instead of some `Solver` and `Algorithm` which are served by `py2dmat`, users can define and use their own components. In this chapter, how to define `Solver` and `Algorithm` and to use them will be described.

9.1 Commons

9.1.1 `py2dmat.Info`

This class treats the input parameters. This has the following four instance variables.

- `base : dict[str, Any]`
  - Parameters for whole program such as the directory where the output will be written.
- `solver : dict[str, Any]`
  - Parameters for `Solver`
- `algorithm : dict[str, Any]`
  - Parameters for `Algorithm`
- `runner : dict[str, Any]`
  - Parameters for `Runner`

An instance of `Info` is initialized by passing a `dict` which has the following four sub dictionaries, `"base"`, `"solver"`, `"algorithm"`, and `"runner"`. Each value will be set to the corresponding field of `Info`.

- About `base`
  - Root directory `root_dir`
    - The default value is `"."` (the current directory).
    - Value of `root_dir` will be converted to an absolute path.
    - The leading ~ will be expanded to the user's home directory.
    - Specifically, the following code is executed

```
p = pathlib.Path(base.get("root_dir", "."))
base["root_dir"] = p.expanduser().absolute()
```

  - Output directory `output_dir`

* The default value is `"."`, that is, the same to `root_dir`
* The leading ~ will be expanded to the user's home directory.
* If a relative path is given, its origin is `root_dir`.
* Specifically, the following code is executed

```
p = pathlib.Path(base.get("work_dir", "."))
p = p.expanduser()
base["work_dir"] = base["root_dir"] / p
```

9.1.2 py2dmat.Message

When `Algorithm` tries to invoke `Solver`, an instance of this class is passed from `Algorithm` to `Solver` via `Runner`.

This has the following three instance variables.

- `x: np.ndarray`
  - Coordinates of a point x to calculate $f(x)$
- `step: int`
  - The index of parameters
  - For example, the index of steps in `exchange` and the ID of parameter in `mapper`.
- `set: int`
  - Which lap it is
  - For example, `min_search` has two laps, the first one is optimization and the second one is recalculation the optimal values for each step.

9.1.3 py2dmat.Runner

`Runner` connects `Algorithm` and `Solver`. The constructor of `Runner` takes `Solver` and `Info`.

`submit(self, message: py2dmat.Message) -> float` method invokes the solver and returns the result. To evaluate `fx = f(x)`, use the following code snippet:

```
message = py2dmat.Message(x, step, set)
fx = runner.submit(message)
```

9.2 Solver

`Solver` is defined as a subclass of `py2dmat.solver.SolverBase`

```
import py2dmat

class Solver(py2dmat.solver.SolverBase):
    ...
```

The following methods should be defined.

- `__init__(self, info: py2dmat.Info)`

  - It is required to call the constructor of the base class.
    - `super().__init__(info)`
  - The constructor of `SolverBase` defines the following instance variables.
    - `self.root_dir: pathlib.Path` : Root directory
      - use `info.base["root_dir"]`
    - `self.output_dir: pathlib.Path` : Output directory
      - use `info.base["output_dir"]`
    - `self.proc_dir: pathlib.Path` : Working directory for each MPI process
      - as `self.output_dir / str(mpirank)`
    - `self.work_dir: pathlib.Path` : Directory where the solver is invoked
      - same to `self.proc_dir`
  - Read the input parameter `info` and save as instance variables.
- `default_run_scheme(self) -> str`
  - Returns the default method to invoke the solver.
  - The followings are available:
    - `subprocess` : run as a process via `subprocess.run`
    - `function` : run as a python function
  - In future, we plans to make a `Solver` support multiple way to invoke.
- `prepare(self, message: py2dmat.Message) -> None`
  - This is called before the solver starts
  - `message` includes an input parameter `x`, convert it to something to be used by the solver
    - e.g., to generate an input file of the solver
- `get_results(self) -> float`
  - This is called after the solver finishes
  - Returns the result of the solver
    - e.g., to retrieve the result from the output file of the solver

One of the following two method should be defined:

- `command(self) -> List[str]`
  - Returns a command to invoke the solver
  - The return value will be transferred to `subprocess.run`
  - This method is nessesary when `default_run_scheme` returns `"subprocess"`
- `function(self) -> Callable[[], None]`
  - Returns a python function to invoke the solver
  - The return value (function) will be called immediately.
  - This method is nessesary when `default_run_scheme` returns `"function"`

9.3 Algorithm

`Algorithm` is defined as a subclass of `py2dmat.algorithm.AlgorithmBase`

```
import py2dmat

class Algorithm(py2dmat.algorithm.AlgorithmBase):
    ...
```

9.3.1 AlgorithmBase

`AlgorithmBase` class offers the following methods

```
- ``__init__(self, info: py2dmat.Info, runner: py2dmat.Runner = None)``
```

- Reads the common parameters from `info` and sets the following instance variables:
  - `self.mpicomm: Optional[MPI.Comm]` : `MPI.COMM_WORLD`
    - When `import mpi4py` fails, this will be `None`.
  - `self.mpisize: int` : the number of MPI processes
    - When `import mpi4py` fails, this will be `1`.
  - `self.mpirank: int` : the rank of this process
    - When `import mpi4py` fails, this will be `0`.
  - `self.rng: np.random.Generator` : pseudo random number generator
    - For details of the seed, please see *the [algorithm] section of the input parameter*
  - `self.dimension: int` : the dimension of the parameter space
  - `self.label_list: List[str]` : the name of each axes of the parameter space
  - `self.root_dir: pathlib.Path` : root directory
    - `info.base["root_dir"]`
  - `self.output_dir: pathlib.Path` : output directory
    - `info.base["root_dir"]`
  - `self.proc_dir: pathlib.Path` : working directory of each process
    - `self.output_dir / str(self.mpirank)`
    - Directory will be made automatically
    - Each process performs an optimization algorithm in this directory
  - `self.timer: dict[str, dict]` : dictionary storing elapsed time
    - Three empty dictinaries, `"prepare"`, `"run"`, and `"post"` will be defined
- `prepare(self) -> None`
  - Prepares the algorithm
  - It should be called before `self.run()` is called
  - It calls `self._prepare()`

- `run(self) -> None`
  - Performs the algorithm
  - Enters into `self.proc_dir`, calls `self._run()`, and returns to the original directory.
- `post(self) -> None`
  - Runs a post process of the algorithm, for example, write the result into files
  - It should be called after `self.run()` is called
  - Enters into `self.output_dir`, calls `self._post()`, and returns to the original directory.
- `main(self) -> None`
  - Calls `prepare`, `run`, and `post`
  - Measures the elapsed times for calling each function, and write them into file
- `_read_param(self, info: py2dmat.Info) -> Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]`
  - Helper method for initializing defining the continuous parameter space
  - Reads `info.algorithm["param"]` and returns the followings:
    - Initial value
    - Lower bound
    - Upper bound
    - Unit
  - For details, see *[algorithm.param] subsection for minsearch*
- `_meshgrid(self, info: py2dmat.Info, split: bool = False) -> Tuple[np.ndarray, np.ndarray]`
  - Helper method for initializing defining the discrete parameter space
  - Reads `info.algorithm["param"]` and returns the followings:
    - `N` points in the `D` dimensinal space as a `NxD` matrix
    - IDs of points as a `N` dimensional vector
  - If `split` is `True`, the set of points is scatterred to MPI processes
  - For details, see *[algorithm.param] subsection for mapper*

9.3.2 `Algorithm`

In `Algorithm`, the following methods should be defined:

- `__init__(self, info: py2dmat.Info, runner: py2dmat.Runner = None)`
  - Please transfer the arguments to the constructor of the base class:
    - `super().__init__(info=info, runner=runner)`
  - Reads `info` and sets information
- `_prepare(self) -> None`
  - Pre process
- `_run(self) -> None`

  - The algorithm itself
  - In this method, you can calculate `f(x)` from a parameter `x` as the following:

```
message = py2dmat.Message(x, step, set)
fx = self.runner.submit(message)
```

- `_post(self) -> None`
  - Post process

9.4 Usage

The following flow solves the optimization problem. The number of flow corresponds the comment in the program example.

1. Define your `Algorithm` and/or `Solver`
   - Of course, classes that `py2dmat` defines are available
2. Prepare the input parameter, `info: py2dmat.Info`
   - Make a dictionary as your favorite way
     - The below example uses a TOML formatted input file for generating a dictionary
3. Instantiate `solver: Solver`, `runner: py2dmat.Runner`, and `algorithm: Algorithm`
4. Invoke `algorithm.main()`

Example:

```
import sys
import toml
import py2dmat

# (1)
class Solver(py2dmat.solver.SolverBase):
    # Define your solver
    ...

class Algorithm(py2dmat.algorithm.AlgorithmBase):
    # Define your algorithm
    ...


file_name = sys.argv[1]

# (2)
info = py2dmat.Info(toml.load(file_name))

# (3)
solver = Solver(info)
runner = py2dmat.Runner(solver, info)
algorithm = Algorithm(info, runner)

# (4)
algorithm.main()
```

CHAPTER

TEN

ACKNOWLEDGEMENTS

The development of 2DMAT was supported by JSPS KAKENHI Grant Number 19H04125 “Unification of computational statistics and measurement technology by massively parallel machine” and “Project for advancement of software usability in materials science” of The Institute for Solid State Physics, The University of Tokyo.

CHAPTER

ELEVEN

CONTACT

- Bug Reports

  Please report all problems and bugs on the github Issues page.

  To resolve bugs early, follow these guidelines when reporting:

  1. Please specify the version of 2DMAT you are using.
  2. If there are problems for installation, please inform us about your operating system and the compiler.
  3. If a problem occurs during execution, enter the input file used for execution and its output.

  Thank you for your cooperation.

- Others

  If you have any questions about your research that are difficult to consult at Issues on GitHub, please send an e-mail to the following address:

  E-mail: `2dmat-dev__at__issp.u-tokyo.ac.jp` (replace _at_ by @)